SEED // Search for Extraordinary Experiences Division
seed.ea.com

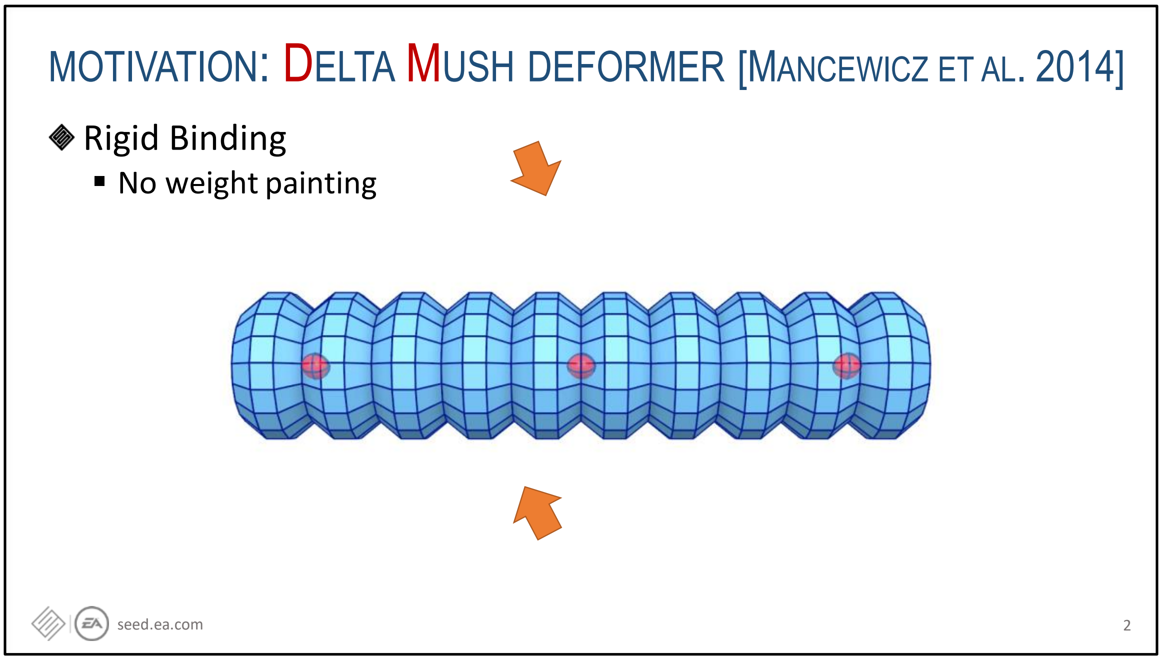
Direct Delta Mush Skinning and Variants

Binh Huy Le JP Lewis*

thrive SIGGRAPH2019 LOS ANGELES • 28 JULY – 1 AUGUST

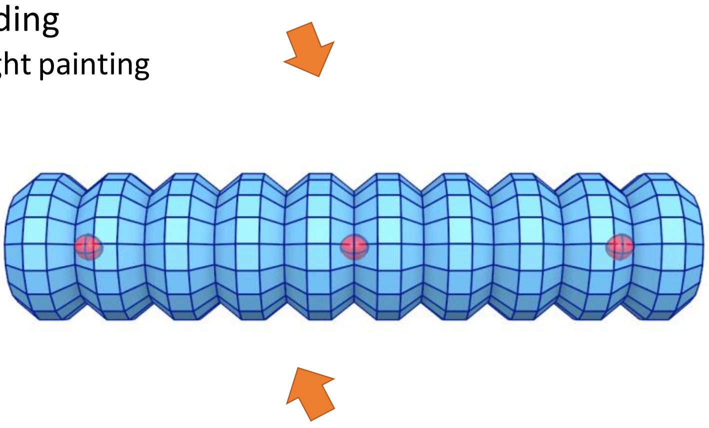
*now with Google AI

(*) JP Lewis was with SEED//Electronic Arts when we worked on this project.



MOTIVATION: DELTA MUSH DEFORMER [MANCEWICZ ET AL. 2014]

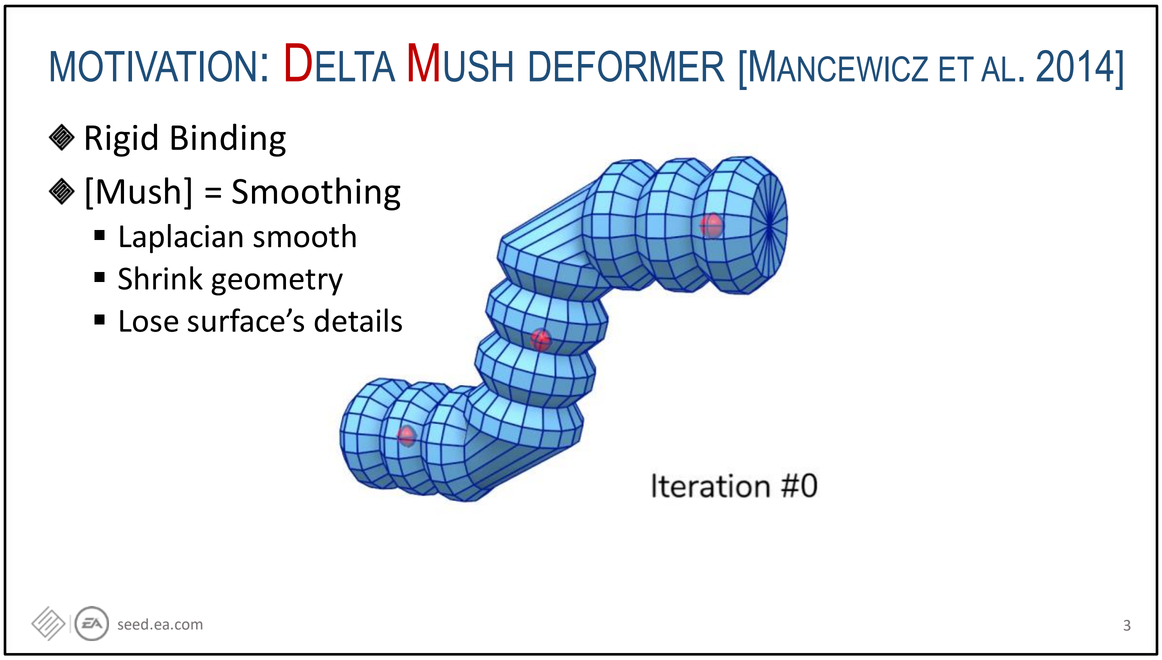
- Rigid Binding
  - No weight painting



seed.ea.com

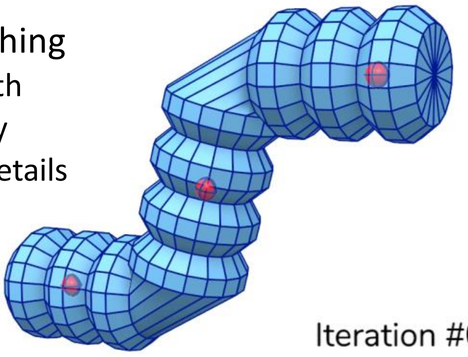
2

So firstly to setup the background, I will talk about Delta Mush.
It is a quite popular tool in industry.
The idea is to reduce the authoring cost by avoiding weight-painting.
For example, if we have 3 joints, drawn in red color here,
[click]
And we will just do the rigid bind, the geometry will break when we animate the joints.



MOTIVATION: DELTA MUSH DEFORMER [MANCEWICZ ET AL. 2014]

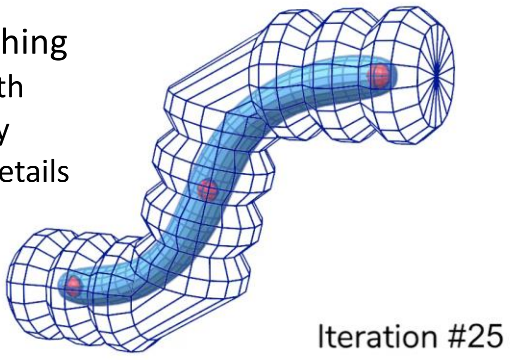
- Rigid Binding
- [Mush] = Smoothing
  - Laplacian smooth
  - Shrink geometry
  - Lose surface's details



To fix this, Delta Mush uses Laplacian smoothing, or "mushing" as named by the original authors.
[next]
Smoothing fixed broken geometry, but it shrinks the mesh and blurs all details.

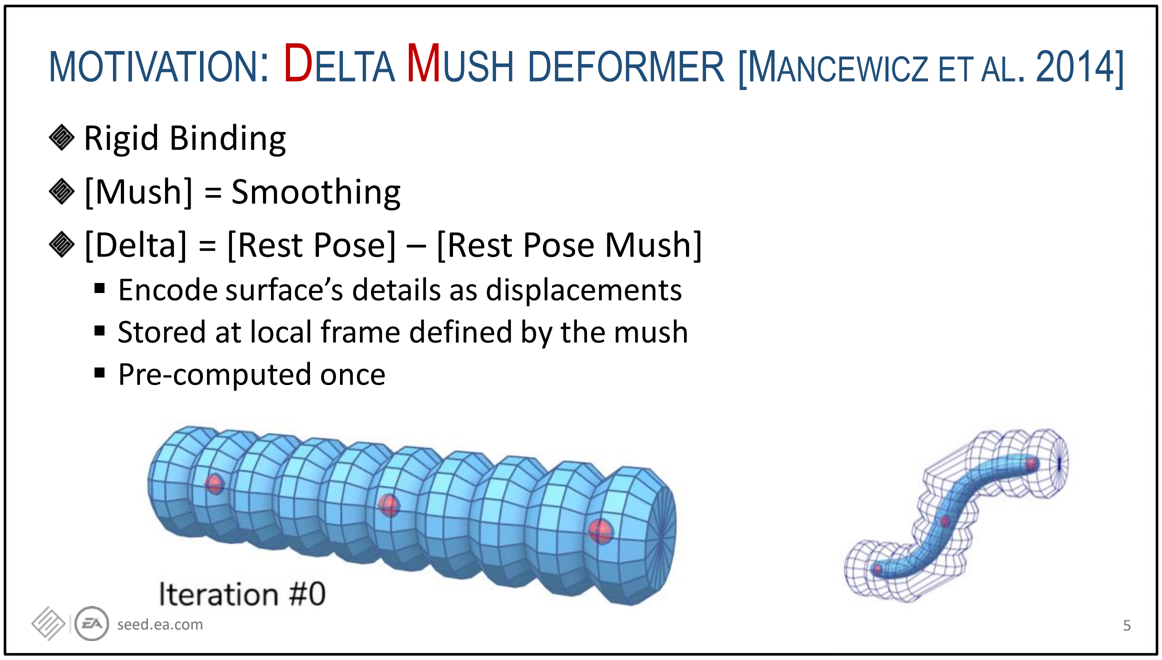
MOTIVATION: DELTA MUSH DEFORMER [MANCEWICZ ET AL. 2014]

- Rigid Binding
- [Mush] = Smoothing
  - Laplacian smooth
  - Shrink geometry
  - Lose surface's details



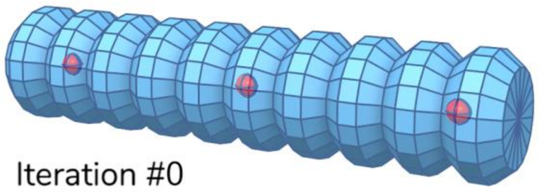
seed.ea.com

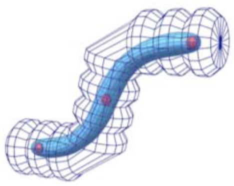
[Transition slide: Just leave move to the side]



MOTIVATION: DELTA MUSH DEFORMER [MANCEWICZ ET AL. 2014]

- Rigid Binding
- [Mush] = Smoothing
- [Delta] = [Rest Pose] – [Rest Pose Mush]
  - Encode surface's details as displacements
  - Stored at local frame defined by the mush
  - Pre-computed once



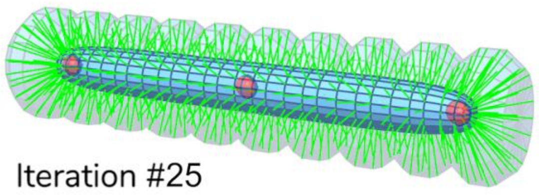


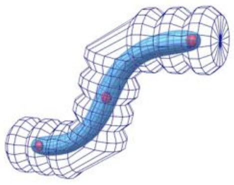
seed.ea.com 5

To recover the details, we compute the "Delta",
[next]
it is the difference in the rest pose between the original geometry and the smooth version stored in the local coordinate frame,
those are green lines in this figure.
And note that we only need to compute the "Delta" once per model, but not at run-time.

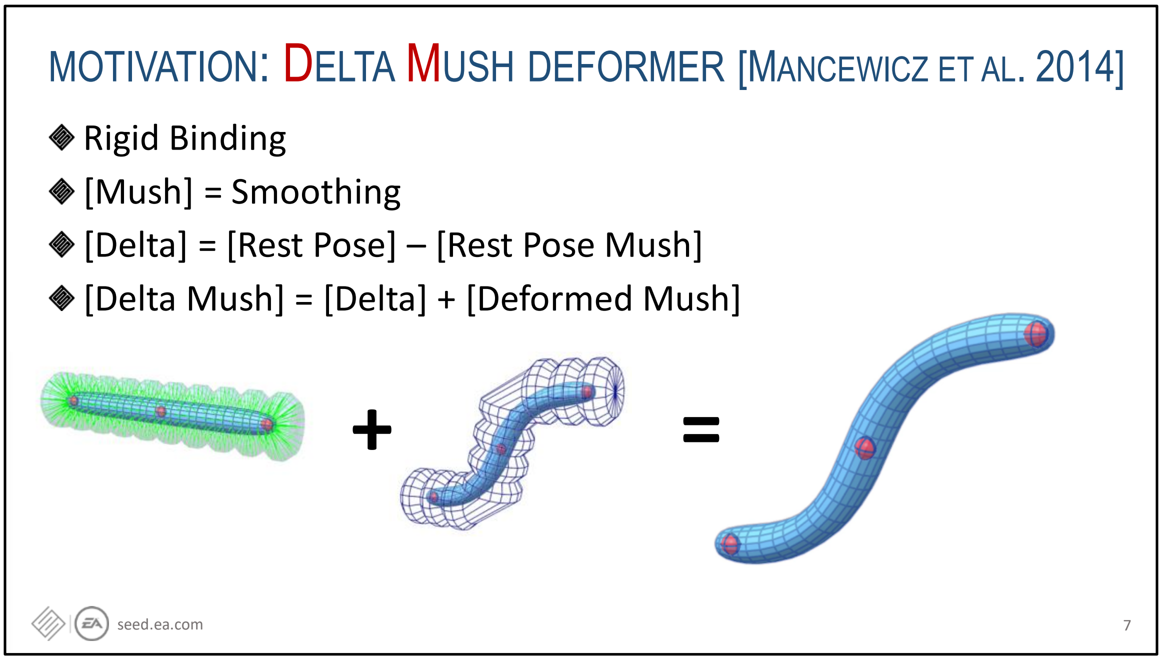
MOTIVATION: DELTA MUSH DEFORMER [MANCEWICZ ET AL. 2014]

- Rigid Binding
- [Mush] = Smoothing
- [Delta] = [Rest Pose] – [Rest Pose Mush]



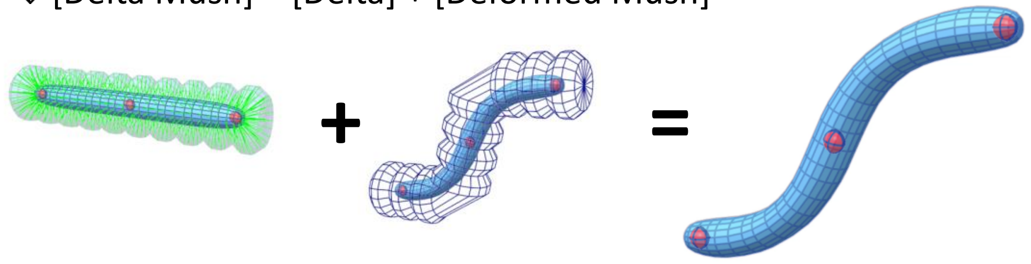


So now we will use the Delta to recover the surface's details of the mush.



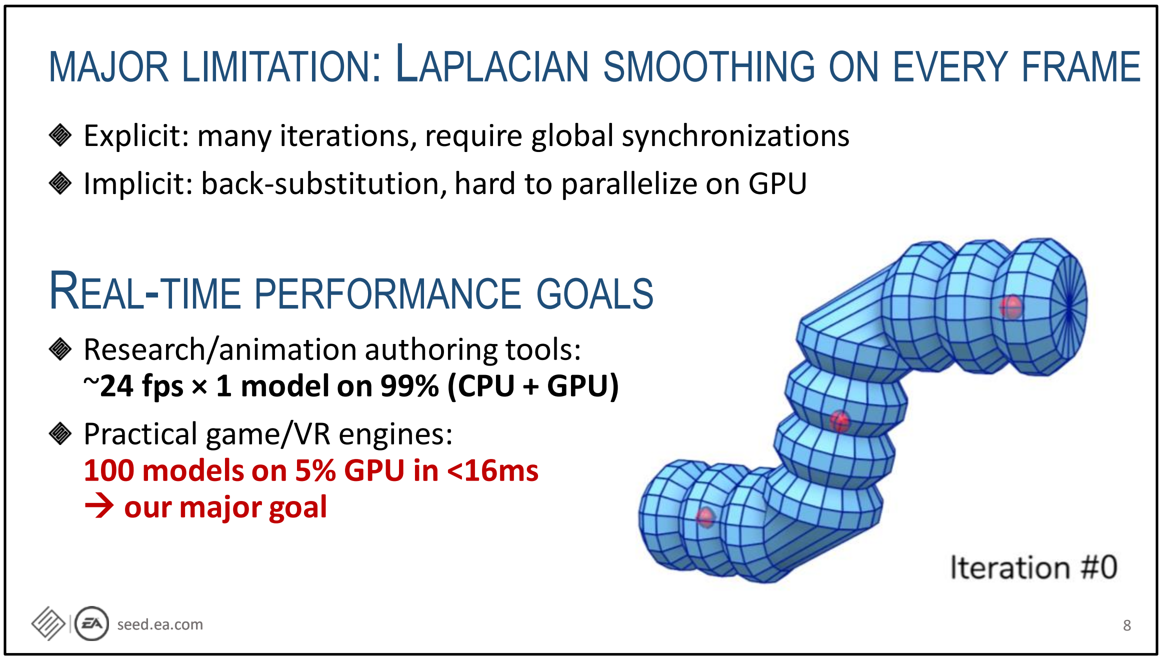
MOTIVATION: DELTA MUSH DEFORMER [MANCEWICZ ET AL. 2014]

- Rigid Binding
- [Mush] = Smoothing
- [Delta] = [Rest Pose] – [Rest Pose Mush]
- [Delta Mush] = [Delta] + [Deformed Mush]



seed.ea.com 7

[next]
when we add the Delta on top the Mush using the local coordinate frame of the deformed mush,
[next]
we have the result with smooth deformation and sharp geometry details.
The great thing about Delta Mush is it produces good quality deformation with very simple concept and setup.

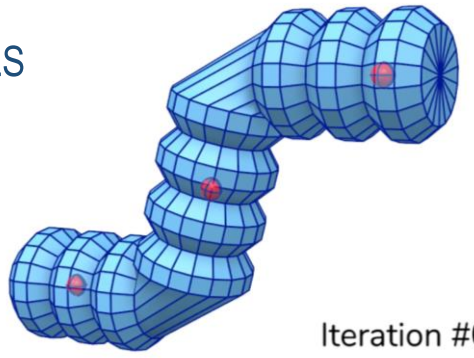


MAJOR LIMITATION: LAPLACIAN SMOOTHING ON EVERY FRAME

- Explicit: many iterations, require global synchronizations
- Implicit: back-substitution, hard to parallelize on GPU

REAL-TIME PERFORMANCE GOALS

- Research/animation authoring tools:
  ~24 fps × 1 model on 99% (CPU + GPU)
- Practical game/VR engines:
  100 models on 5% GPU in <16ms
  → our major goal

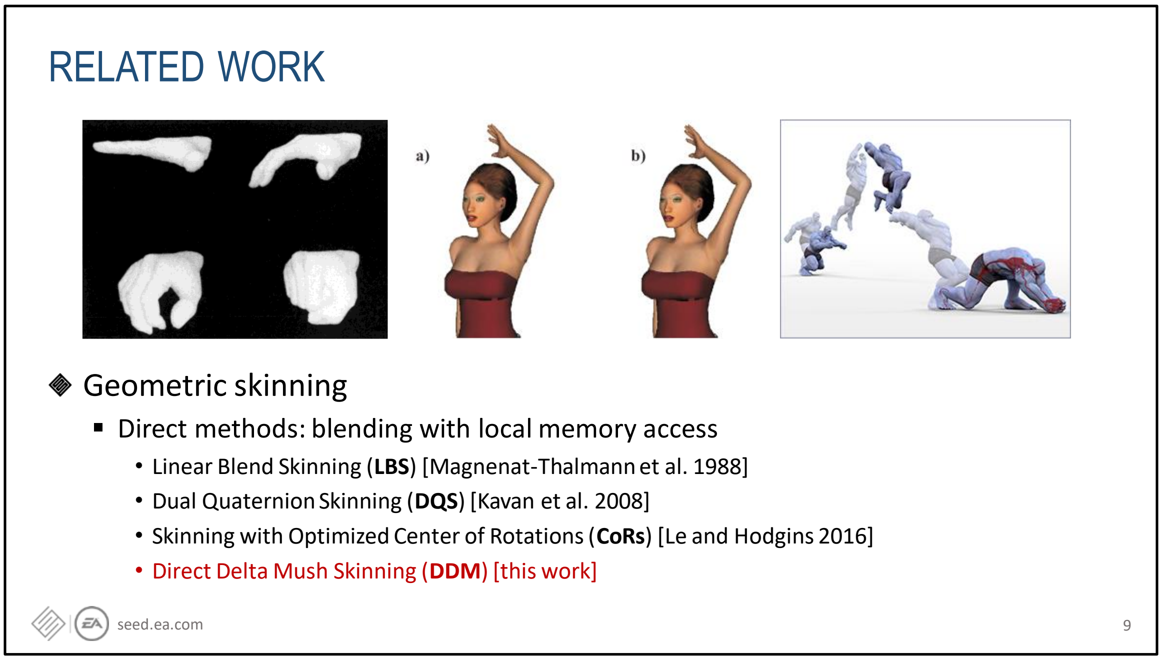


seed.ea.com 8

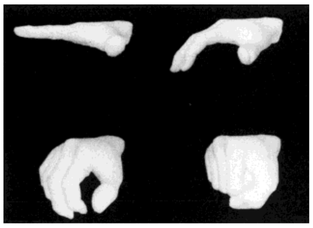
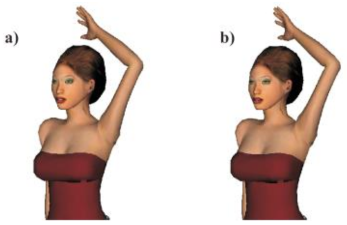
But in term of computational complexity, it is not so nice.
The Laplacian smoothing is an iterative algorithm.
For each animation frame, many iterations need to run sequentially with synchronizations in between them.
Using implicit Laplacian can reduce the number of iterations
But the trade off is each iteration need to solve large scale linear system. And even with pre-factorization, the back-substitution step is hard to parallel on GPU.

So even Delta Mush can easily reach the real-time performance for small scale research and authoring tools,
it is not suitable for large scale and high-performance game and VR engines.

And our goal for this project is to make Delta Mush to be a direct skinning method.



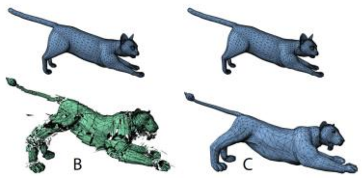
RELATED WORK

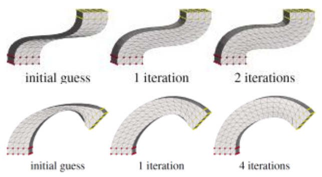



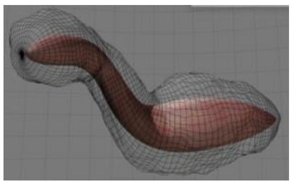
- Geometric skinning
  - Direct methods: blending with local memory access
    - Linear Blend Skinning (**LBS**) [Magnenat-Thalmann et al. 1988]
    - Dual Quaternion Skinning (**DQS**) [Kavan et al. 2008]
    - Skinning with Optimized Center of Rotations (**CoRs**) [Le and Hodgins 2016]
    - Direct Delta Mush Skinning (**DDM**) [this work]

And here is a quick review of the landscape.
Our work shares the similar computing architecture with other direct methods, which typically can compute all vertex transformations in parallel and the computation on each vertex only need to access small local memory.
This is suitable for implementation on GPU shaders.

RELATED WORK



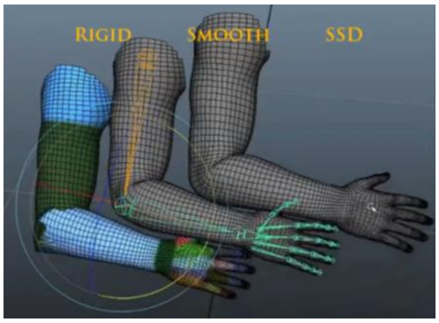




- Geometric skinning
  - Indirect methods: global large-scale solver, iterative with sync
    - Poisson stitching [Sumner and Popović 2004, Wang et al. 2007]
    - As-rigid-as-possible [Sorkine and Alexa 2007, Jacobson et al. 2012]
    - Delta Mush (**DM**) [Mancewicz et al. 2014]

Indirect methods can optimize some global energy term
so they typically offer higher quality.
But the computation costs are more costly,
either with a large-scale linear solver like Poison stitching,
or with many iterations and synchronizations like ARAP or DM.

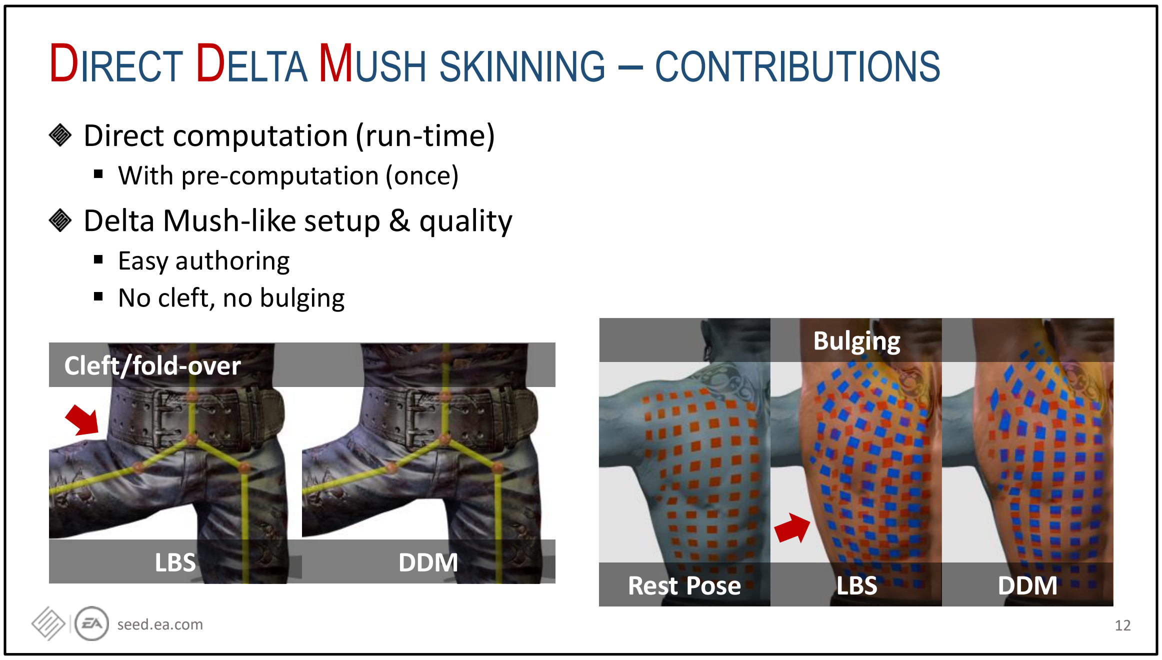
RELATED WORK



- ◈ Baking/skinning decomposition
  - DM to LBS: Hans Godard's Skinning Converter [2015], Maya, etc.

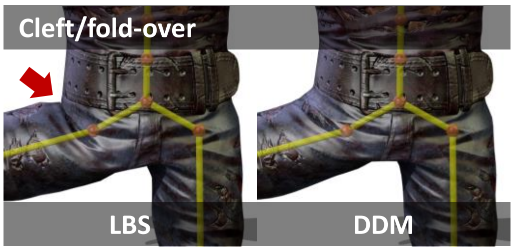
(main inspiration of our work)

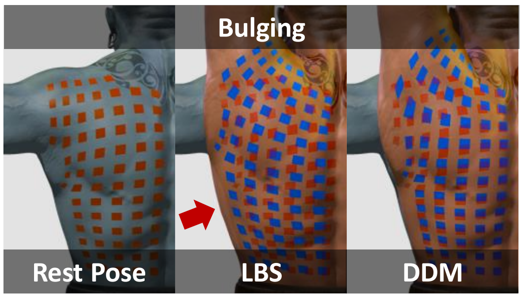
It's also worth to mention that in industry, there is a practice of learning LBS skinning weights from example poses, which is a data fitting problem.
Example poses may be modeled by artist or generated by simulation.
And there was also an idea of using DM to generate example poses.
That was nice because DM is easy to setup, but would be nicer if we don't even need to do data fitting.



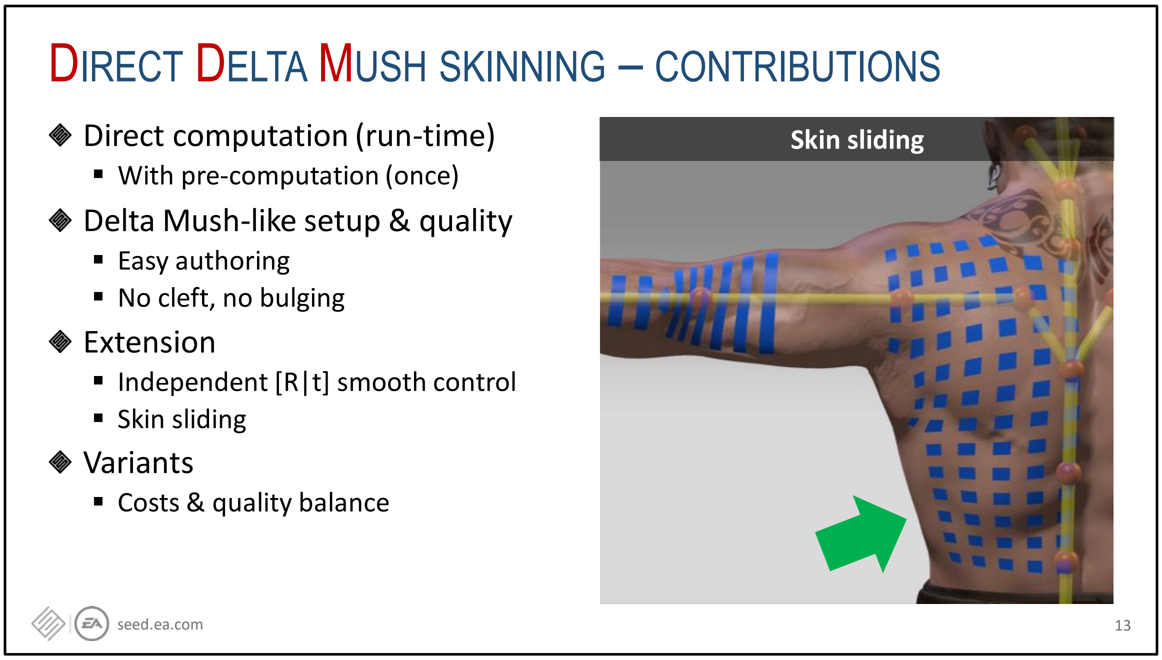
DIRECT DELTA MUSH SKINNING – CONTRIBUTIONS

- Direct computation (run-time)
  - With pre-computation (once)
- Delta Mush-like setup & quality
  - Easy authoring
  - No cleft, no bulging



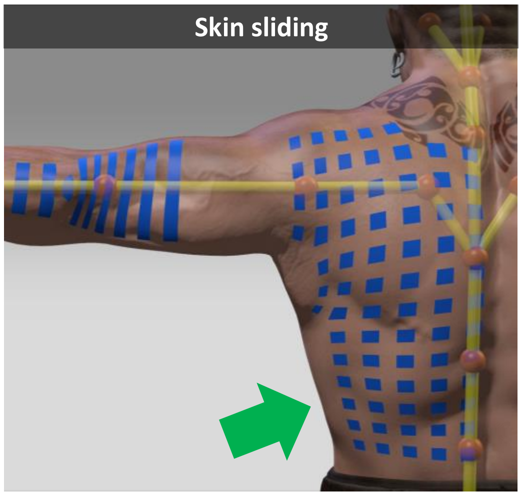


With that inspiration, we reworked the math and made a direct model that respects all the goods of the original Delta Mush,
which means high quality deformation with no weight painting.



DIRECT DELTA MUSH SKINNING – CONTRIBUTIONS

- Direct computation (run-time)
  - With pre-computation (once)
- Delta Mush-like setup & quality
  - Easy authoring
  - No cleft, no bulging
- Extension
  - Independent [R|t] smooth control
  - Skin sliding
- Variants
  - Costs & quality balance

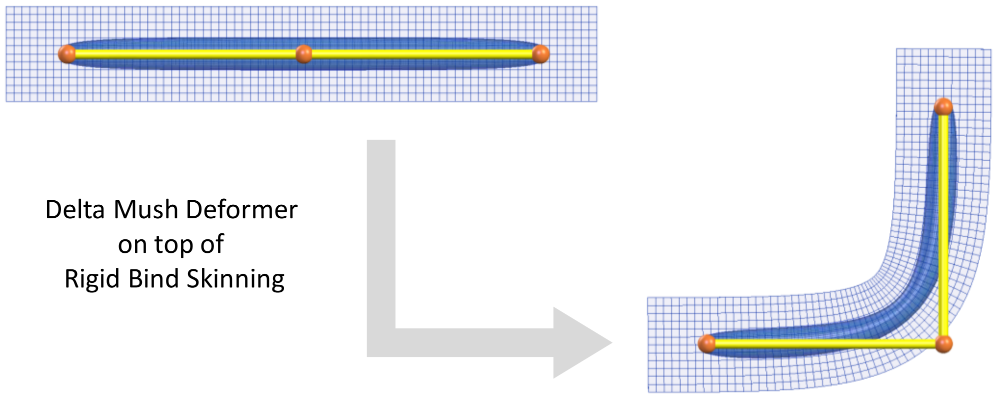


seed.ea.com

13

We even extended the original model to improve the quality.
Here we show the skin sliding effect produced by setting different smoothness amounts of rotation and translation.
To our knowledge, this is the first direct model can do skin sliding.
[click]
And finally, we offer different variants to balance computing, storage costs and deformation quality

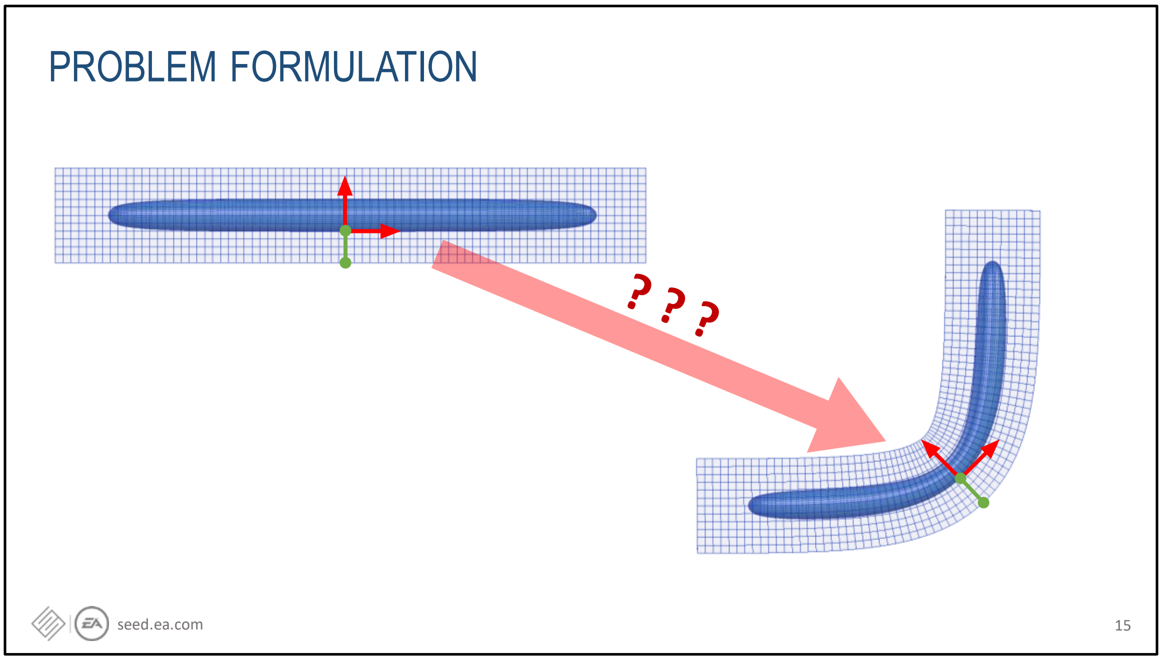
PROBLEM FORMULATION



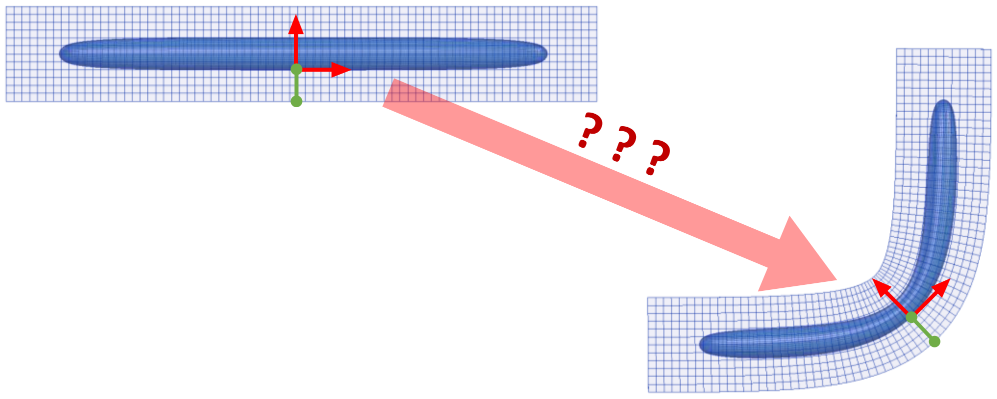
Delta Mush Deformer
on top of
Rigid Bind Skinning

seed.ea.com

Now I will present the main idea of our model.
Suppose that we deform a bar using Delta Mush where the top left figure here shows the rest pose and the bottom right shows the deformed pose.
The bold blue at the core of each bar shows the corresponding mush.
And the deformation is driven by two rigid bones shown in yellow.
Our setup here is delta mush on top of rigid binding.



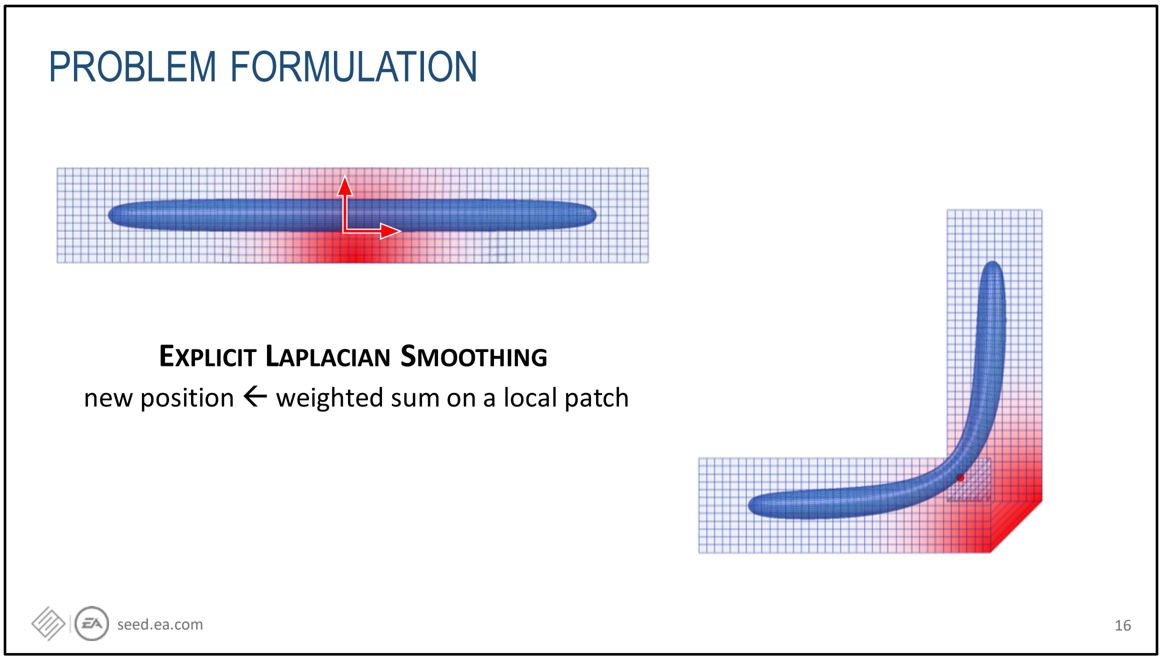
PROBLEM FORMULATION



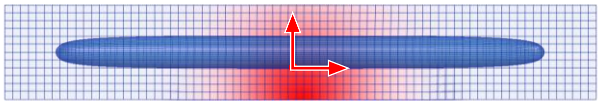
seed.ea.com

15

For each vertex, Delta Mush needs to define a local coordinate frame to store the delta.
So if I visualize the local frame by red arrows. and the delta in green here should be invariant in the red local frame.
And that means the local transformation to deform the mush, which is this [Click] is also the local transformation to deform the final geometry, which is this [Click]

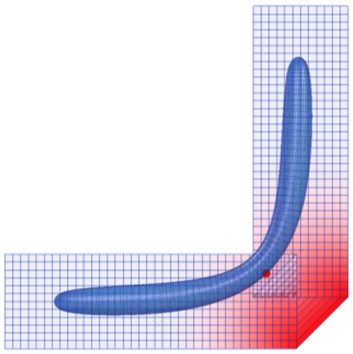


PROBLEM FORMULATION



Explicit Laplacian Smoothing

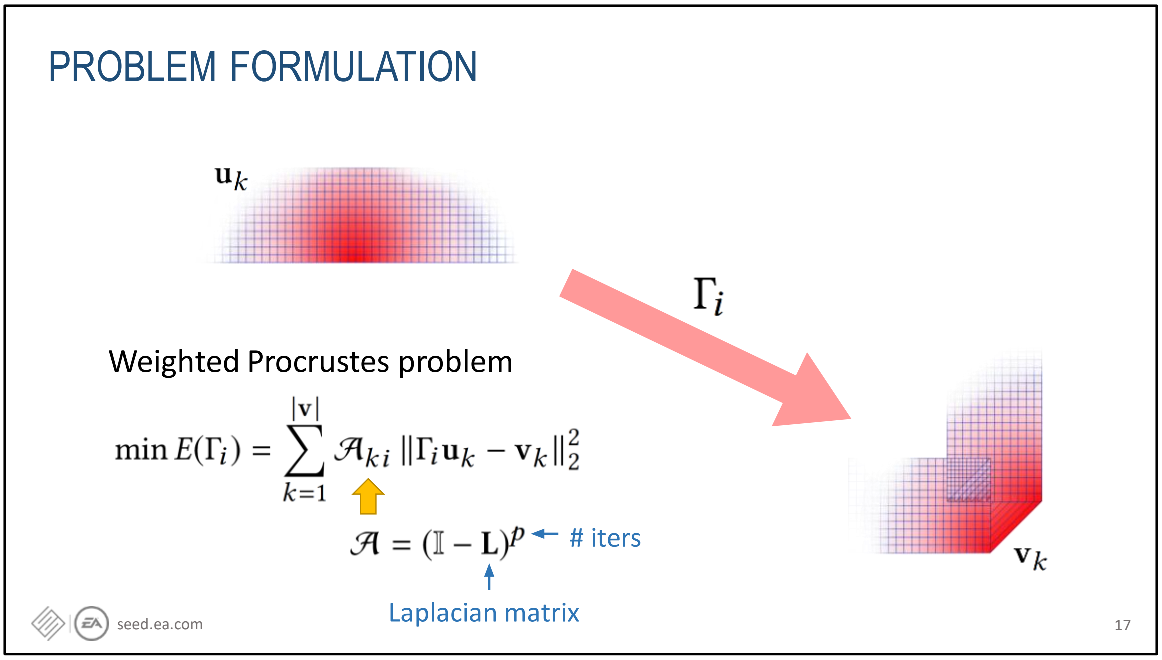
new position ← weighted sum on a local patch



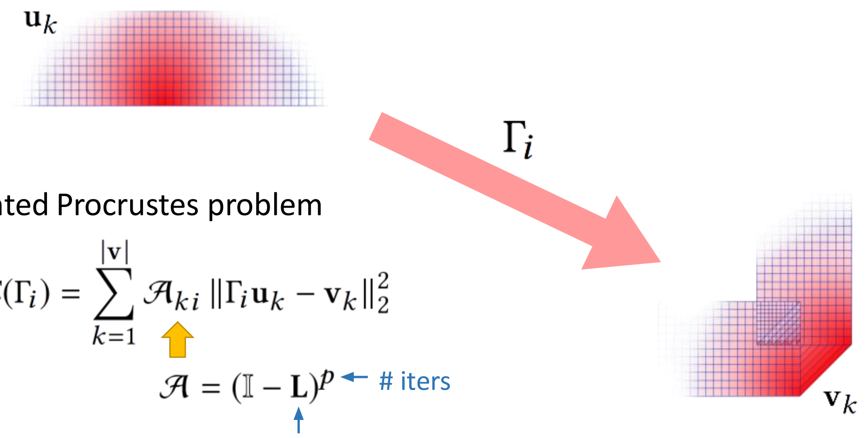
seed.ea.com 16

To solve for the vertex transformation of the mush, we realize that when doing Laplacian smoothing,
the final position of each vertex will be the weighted sum of original vertices on a local patch.
In this figure, we show the patch in red with the smooth fall off represent the weights.

And again, we are trying to find this transformation.
[click]
That will be the same as this transformation of the whole patch.
[click]



PROBLEM FORMULATION



Weighted Procrustes problem

$$\min E(\Gamma_i) = \sum_{k=1}^{|\mathbf{v}|} \mathcal{A}_{ki} \, \|\Gamma_i \mathbf{u}_k - \mathbf{v}_k\|_2^2$$

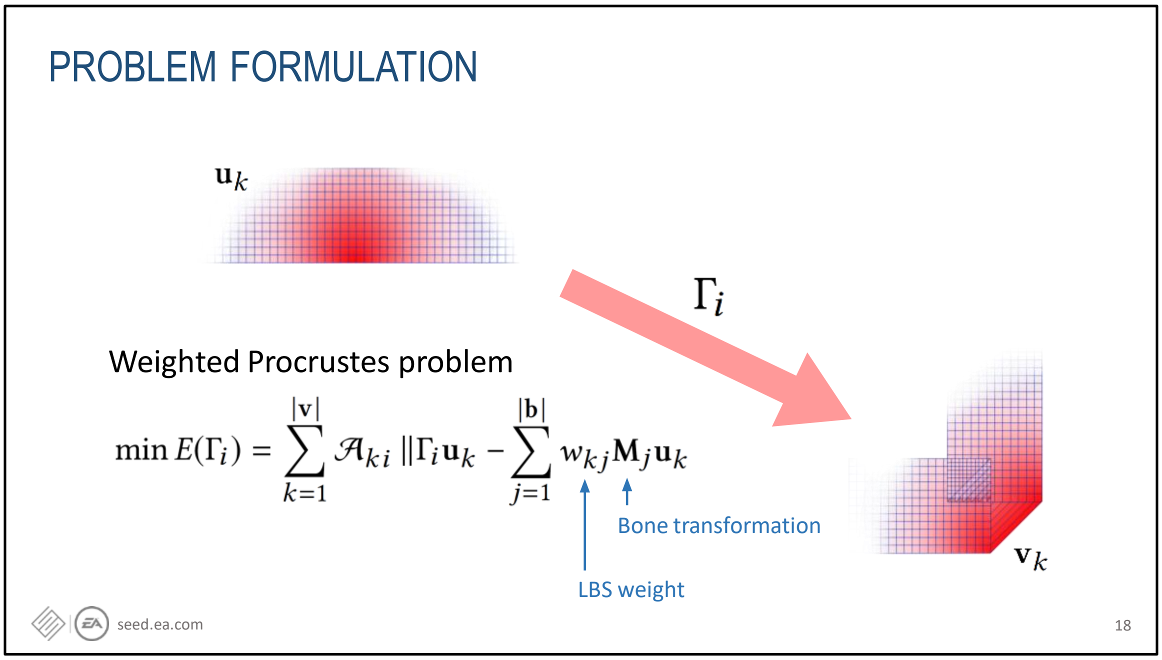
$$\mathcal{A} = (\mathbb{I} - \mathbf{L})^p \leftarrow \text{\# iters}$$

Laplacian matrix

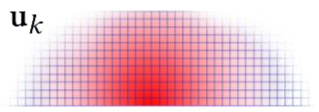
seed.ea.com

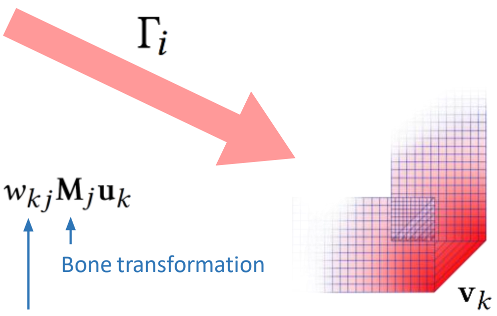
17

This is a weighted Procrustes problem where we find the best rigid transformation by minimizing the sum of squared differences between two set vertices with applying transformation on one set.
In this formulation, the weights of the local patch A sub ki here is defined by the Laplacian matrix L applying p times.
And I is the identity matrix.



PROBLEM FORMULATION





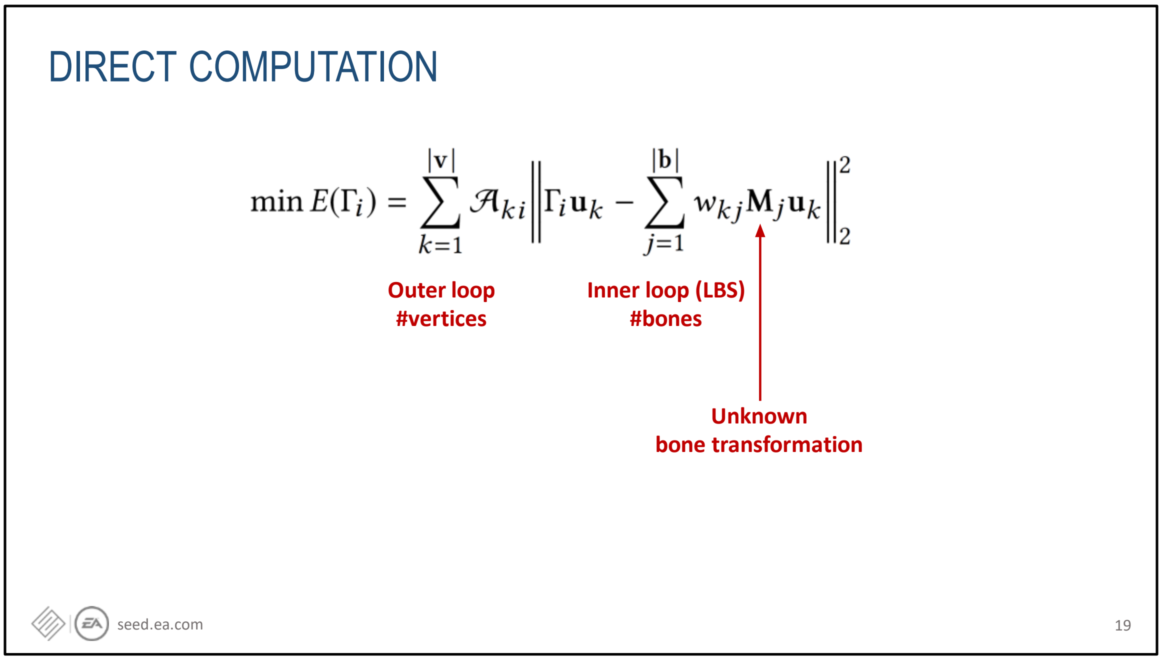
Weighted Procrustes problem

$$\min E(\Gamma_i) = \sum_{k=1}^{|\mathbf{v}|} \mathcal{A}_{ki} \left\| \Gamma_i \mathbf{u}_k - \sum_{j=1}^{|\mathbf{b}|} \underbrace{w_{kj}}_{\text{LBS weight}} \underbrace{\mathbf{M}_j}_{\text{Bone transformation}} \mathbf{u}_k \right.$$

seed.ea.com

18

We can rewrite the objective function by replacing the articulated pose by the Linear blend skinning formula where we can use 0/1 skinning weights.



DIRECT COMPUTATION

$$\min E(\Gamma_i) = \sum_{k=1}^{|\mathbf{v}|} \mathcal{A}_{ki} \left\| \Gamma_i \mathbf{u}_k - \sum_{j=1}^{|\mathbf{b}|} w_{kj} \mathbf{M}_j \mathbf{u}_k \right\|_2^2$$

Outer loop
#vertices

Inner loop (LBS)
#bones

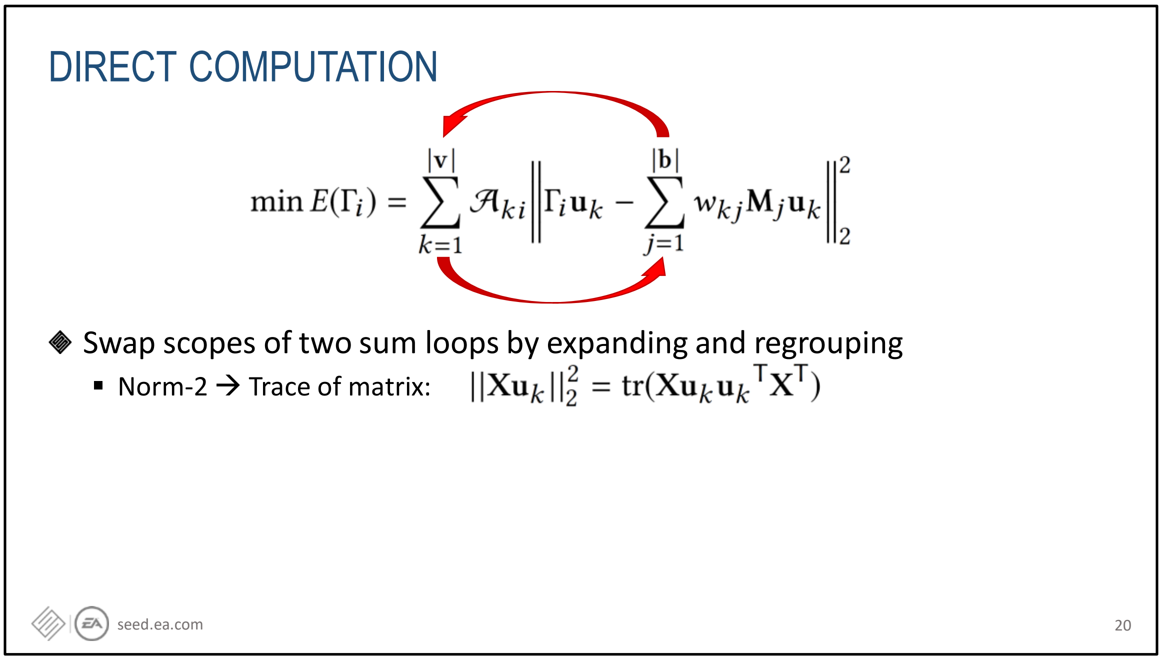
Unknown
bone transformation

seed.ea.com

19

The objective function has two nested sums, you can think of when computing it, you need two nested for loops:
- The outer loop iterates thru all vertices
- And the inner loop iterates thru all bone transformations.

At the inner loop, the bone transformations are unknown before animation,
so with traditional Delta Mush, we need to compute this loop for every frame at runtime.



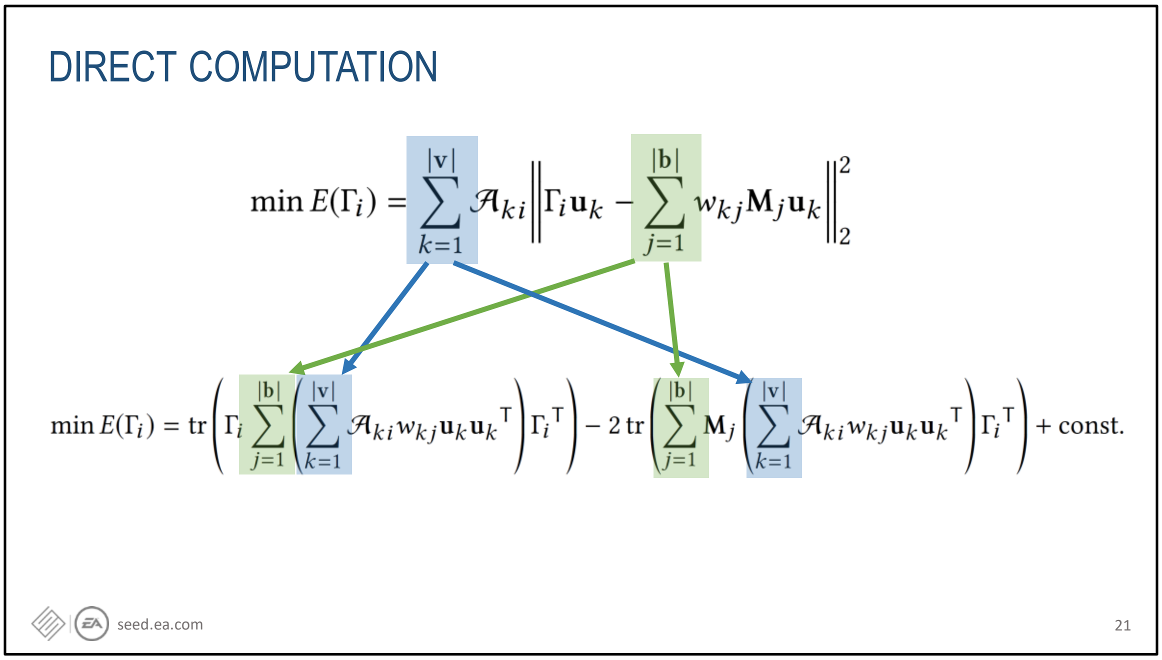
DIRECT COMPUTATION

$$\min E(\Gamma_i) = \sum_{k=1}^{|\mathbf{v}|} \mathcal{A}_{ki} \left\| \Gamma_i \mathbf{u}_k - \sum_{j=1}^{|\mathbf{b}|} w_{kj} \mathbf{M}_j \mathbf{u}_k \right\|_2^2$$

- Swap scopes of two sum loops by expanding and regrouping
  - Norm-2 → Trace of matrix: $||\mathbf{X}\mathbf{u}_k||_2^2 = \mathrm{tr}(\mathbf{X}\mathbf{u}_k {\mathbf{u}_k}^\mathsf{T} \mathbf{X}^\mathsf{T})$

seed.ea.com 20

Instead of doing that, we try to rewrite this expression so that we can swap scopes of two sum loops.
We expand the nested sums and regrouping the terms by the bone transformation M sub j,
Here we use a small trick of rewriting the norm 2 by the trace of the matrix.



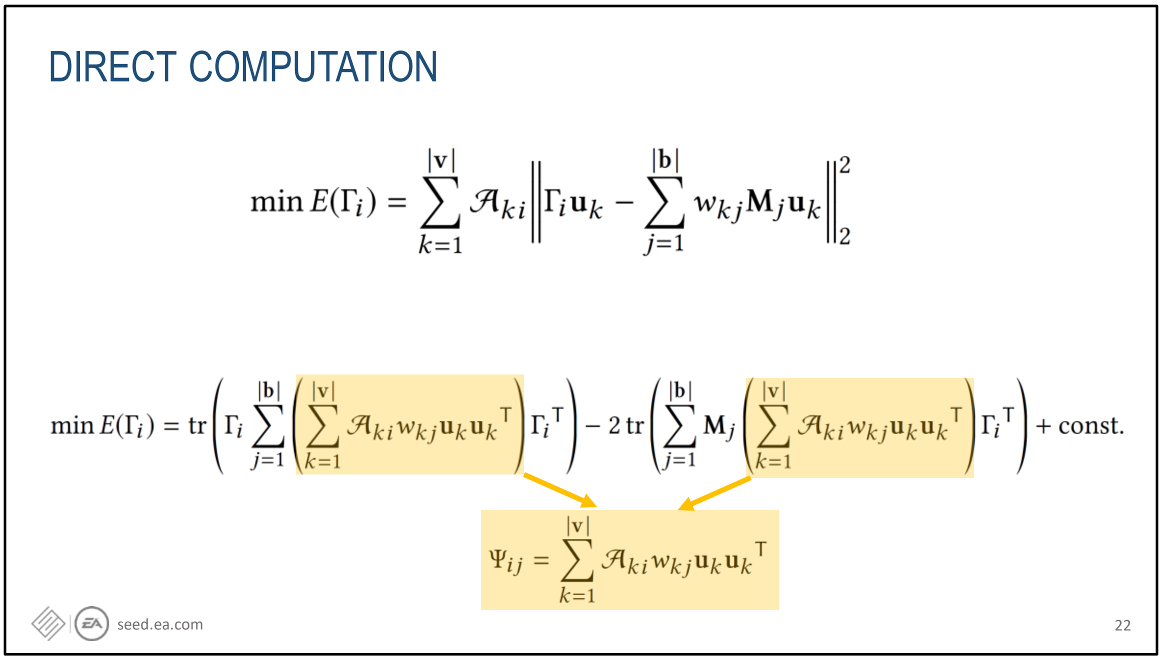
DIRECT COMPUTATION

$$\min E(\Gamma_i) = \sum_{k=1}^{|\mathbf{v}|} \mathcal{A}_{ki} \left\| \Gamma_i \mathbf{u}_k - \sum_{j=1}^{|\mathbf{b}|} w_{kj} \mathbf{M}_j \mathbf{u}_k \right\|_2^2$$

$$\min E(\Gamma_i) = \operatorname{tr}\left(\Gamma_i \sum_{j=1}^{|\mathbf{b}|}\left(\sum_{k=1}^{|\mathbf{v}|} \mathcal{A}_{ki} w_{kj} \mathbf{u}_k {\mathbf{u}_k}^\top\right){\Gamma_i}^\top\right) - 2\operatorname{tr}\left(\sum_{j=1}^{|\mathbf{b}|}\mathbf{M}_j\left(\sum_{k=1}^{|\mathbf{v}|}\mathcal{A}_{ki} w_{kj}\mathbf{u}_k{\mathbf{u}_k}^\top\right){\Gamma_i}^\top\right) + \text{const.}$$

seed.ea.com 21

With expanding and regrouping here is what we ended up with. Now the two sum loops have been swapped.
(please don’t read the equations, just look at the color blocks).



DIRECT COMPUTATION

$$\min E(\Gamma_i) = \sum_{k=1}^{|\mathbf{v}|} \mathcal{A}_{ki} \left\| \Gamma_i \mathbf{u}_k - \sum_{j=1}^{|\mathbf{b}|} w_{kj} \mathbf{M}_j \mathbf{u}_k \right\|_2^2$$

$$\min E(\Gamma_i) = \operatorname{tr}\left(\Gamma_i \sum_{j=1}^{|\mathbf{b}|} \left(\sum_{k=1}^{|\mathbf{v}|} \mathcal{A}_{ki} w_{kj} \mathbf{u}_k {\mathbf{u}_k}^\top \right) {\Gamma_i}^\top \right) - 2 \operatorname{tr}\left(\sum_{j=1}^{|\mathbf{b}|} \mathbf{M}_j \left(\sum_{k=1}^{|\mathbf{v}|} \mathcal{A}_{ki} w_{kj} \mathbf{u}_k {\mathbf{u}_k}^\top \right) {\Gamma_i}^\top \right) + \text{const.}$$

$$\Psi_{ij} = \sum_{k=1}^{|\mathbf{v}|} \mathcal{A}_{ki} w_{kj} \mathbf{u}_k {\mathbf{u}_k}^\top$$

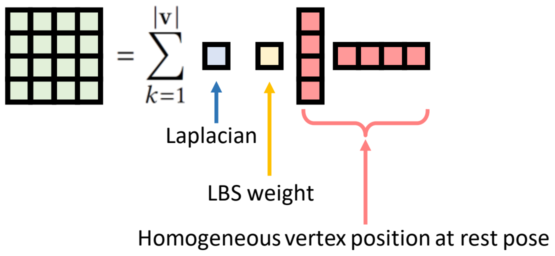
seed.ea.com

Another thing you might notice is that by regrouping the sums, now these two inner sum loops are the same and their values are constant regardless of all bone transformations.
That means we can precompute and cache these.

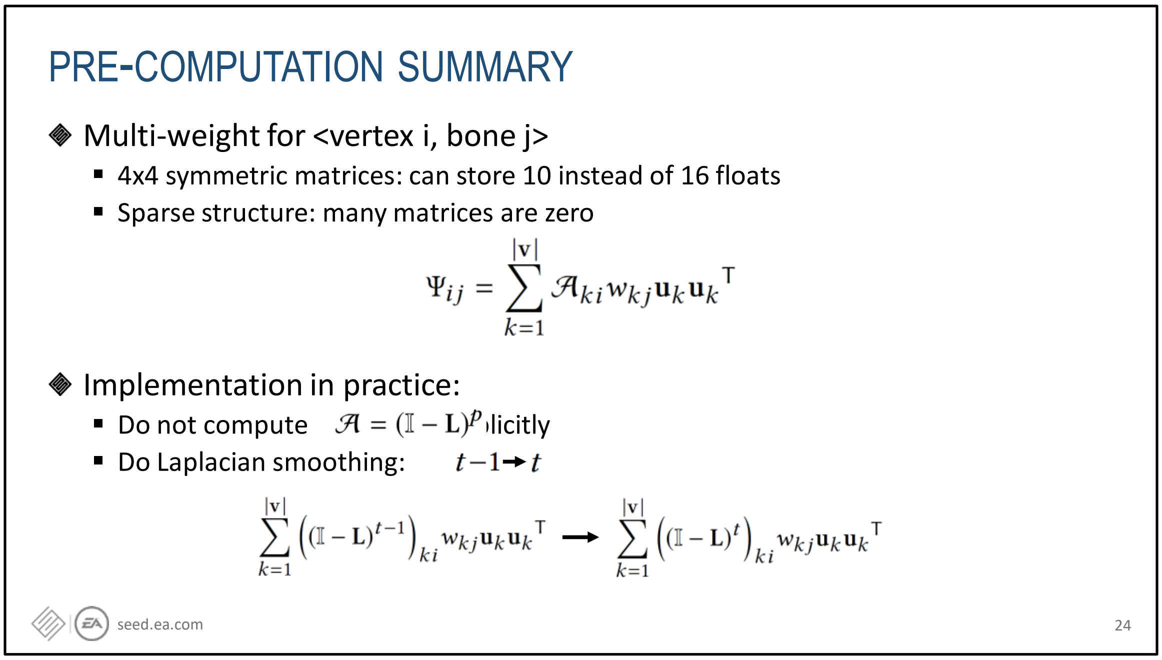
PRE-COMPUTATION SUMMARY

- Multi-weight for <vertex i, bone j>
  - 4x4 symmetric matrices: can store 10 instead of 16 floats
  - Sparse structure: many matrices are zero

$$\Psi_{ij} = \sum_{k=1}^{|\mathbf{v}|} \mathcal{A}_{ki} w_{kj} \mathbf{u}_k \mathbf{u}_k^{\mathsf{T}}$$


$=\sum_{k=1}^{|\mathbf{v}|}$
Laplacian
LBS weight
Homogeneous vertex position at rest pose

Semantically, we found that the terms we precomputed are similar to multi weights.
We denote Psi sub ij as the weight to relate vertex i and bone j.
At the bottom here is the visualization of how they are formed. The Laplacian weight A sub ki and the LBS weight w sub kj here are scalars.
So the final matrix Psi are weighted sum of all outer products u times u transpose.
And those outer products are symmetric matrices.
Therefor Psi are also symmetric, and we can save some storage.
We also note that the sparseness structure of these multi weights are also very similar to the regular scalar skinning weights. That means there will be many zero multi weight matrices.



PRE-COMPUTATION SUMMARY

- Multi-weight for <vertex i, bone j>
  - 4x4 symmetric matrices: can store 10 instead of 16 floats
  - Sparse structure: many matrices are zero

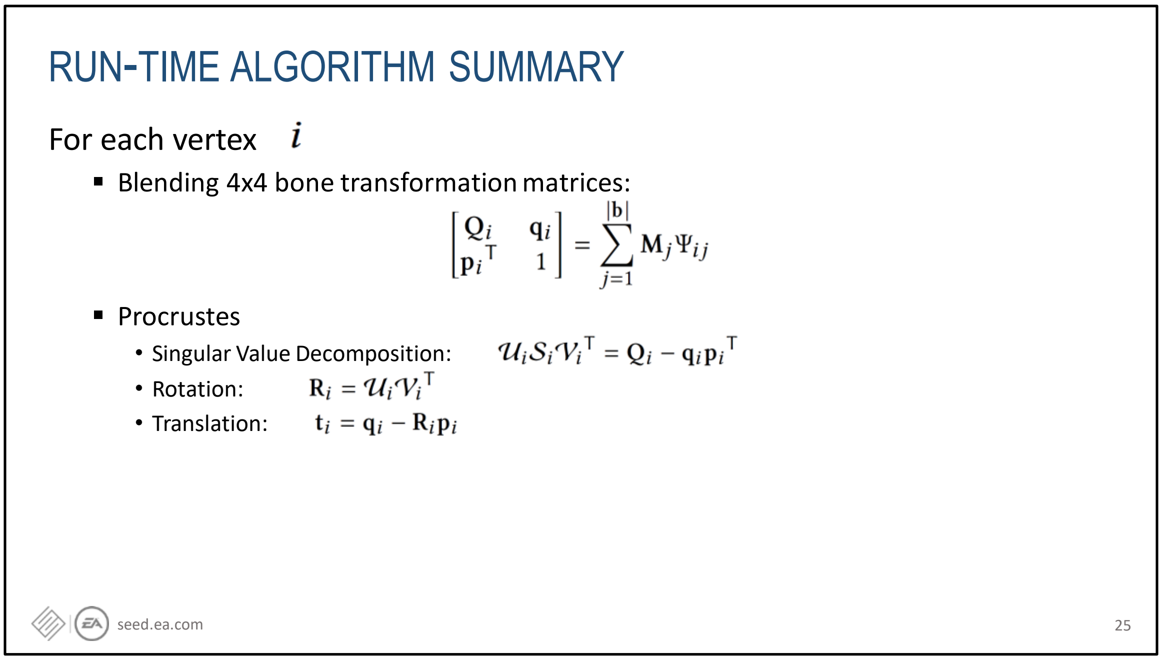
$$\Psi_{ij} = \sum_{k=1}^{|\mathbf{v}|} \mathcal{A}_{ki} w_{kj} \mathbf{u}_k \mathbf{u}_k^{\mathsf{T}}$$

- Implementation in practice:
  - Do not compute $\mathcal{A} = (\mathbb{I} - \mathbf{L})^p$ licitly
  - Do Laplacian smoothing: $t-1 \rightarrow t$

$$\sum_{k=1}^{|\mathbf{v}|} \left((\mathbb{I} - \mathbf{L})^{t-1}\right)_{ki} w_{kj} \mathbf{u}_k \mathbf{u}_k^{\mathsf{T}} \longrightarrow \sum_{k=1}^{|\mathbf{v}|} \left((\mathbb{I} - \mathbf{L})^{t}\right)_{ki} w_{kj} \mathbf{u}_k \mathbf{u}_k^{\mathsf{T}}$$

seed.ea.com 24

And we also note that in practice, we don't need to compute the matrix A here explicitly, this is a square matrix with size of number of vertices and not so sparse so computing it would be very expensive.
Instead, we do Laplacian smoothing to raise the exponential p one by one. Which means we use the Psi matrices at p = t-1 to compute new Psi matrices at p = t. This is equivalent to do Laplacian smoothing on these Psi matrices with some matrix reshaping.



RUN-TIME ALGORITHM SUMMARY

For each vertex i

- Blending 4x4 bone transformation matrices:

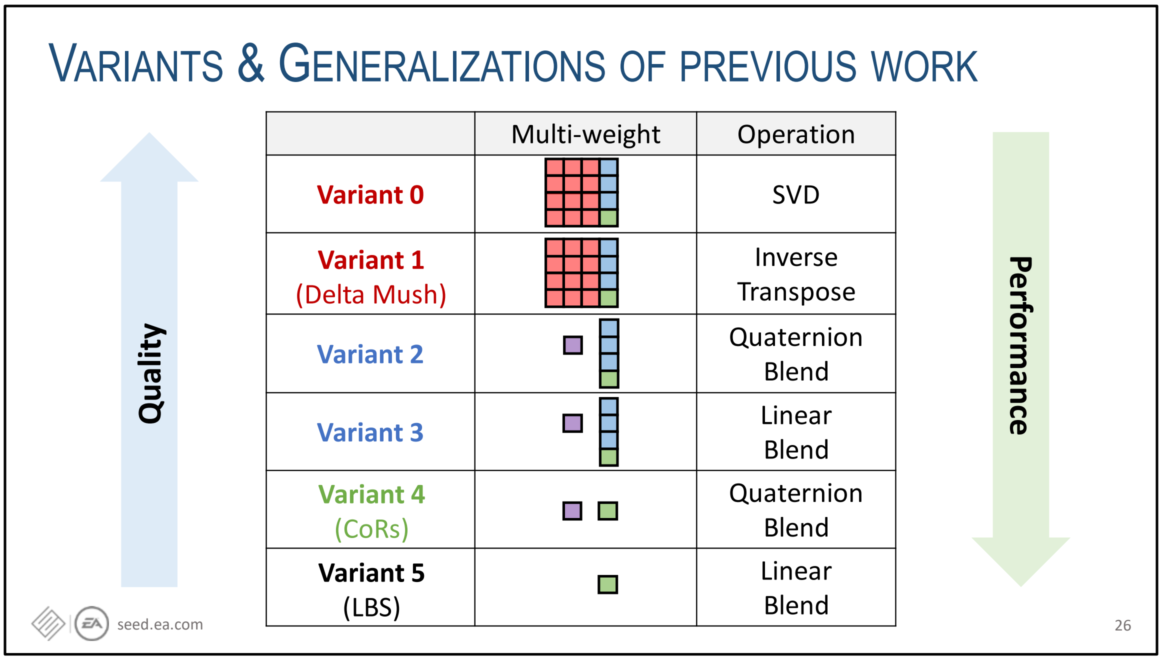
$$\begin{bmatrix} \mathbf{Q}_i & \mathbf{q}_i \\ \mathbf{p}_i^\top & 1 \end{bmatrix} = \sum_{j=1}^{|\mathbf{b}|} \mathbf{M}_j \Psi_{ij}$$

- Procrustes
  - Singular Value Decomposition: $\mathcal{U}_i \mathcal{S}_i \mathcal{V}_i^\top = \mathbf{Q}_i - \mathbf{q}_i \mathbf{p}_i^\top$
  - Rotation: $\mathbf{R}_i = \mathcal{U}_i \mathcal{V}_i^\top$
  - Translation: $\mathbf{t}_i = \mathbf{q}_i - \mathbf{R}_i \mathbf{p}_i$

seed.ea.com 25

And once we have the pre-computed multi-weight matrices Psi, the run-time algorithm is strait forward.
We need to blend input bone transformation matrices M sub j with weights Psi sub ij.
The result is one 4x4 matrix and we can use its sub-blocks Q and p here to solve the Procrustes problem.

Here we use SVD to solve for the rotation matrix R sub i and then use rotation to compute translation vector t sub i.
This algorithm will give the highest deformation quality, however, it could be a bit expensive depending on the implementation of 4x4 matrix operators and the 3x3 SVD.



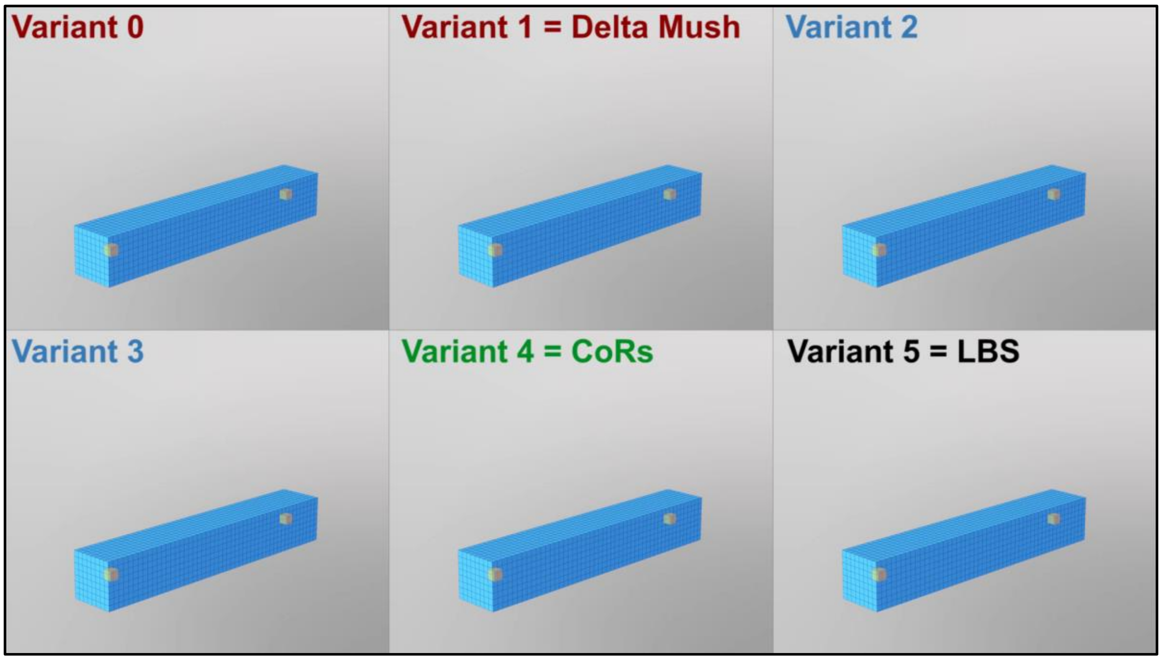
Variants & Generalizations of Previous Work

Quality

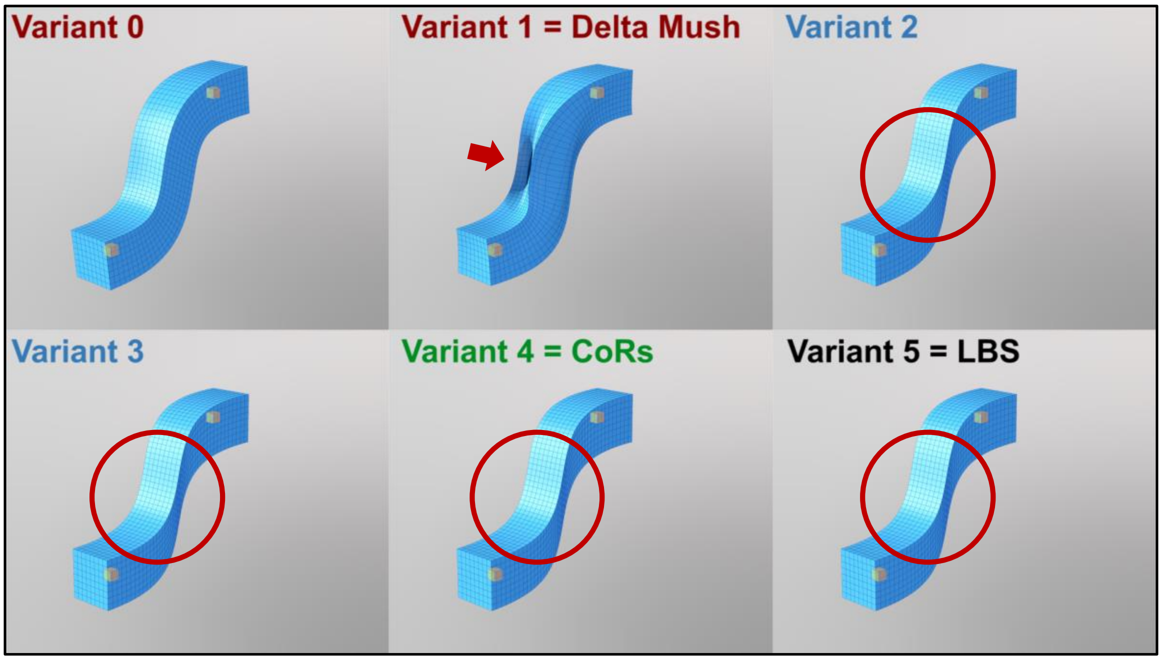
	Multi-weight	Operation
Variant 0	🟥🟥🟥🟦 🟥🟥🟥🟦 🟥🟥🟥🟦 🟥🟥🟥🟩	SVD
Variant 1 (Delta Mush)	🟥🟥🟥🟦 🟥🟥🟥🟦 🟥🟥🟥🟦 🟥🟥🟥🟩	Inverse Transpose
Variant 2	🟪 🟦 🟦 🟦 🟩	Quaternion Blend
Variant 3	🟪 🟦 🟦 🟦 🟩	Linear Blend
Variant 4 (CoRs)	🟪 🟩	Quaternion Blend
Variant 5 (LBS)	🟩	Linear Blend

Performance

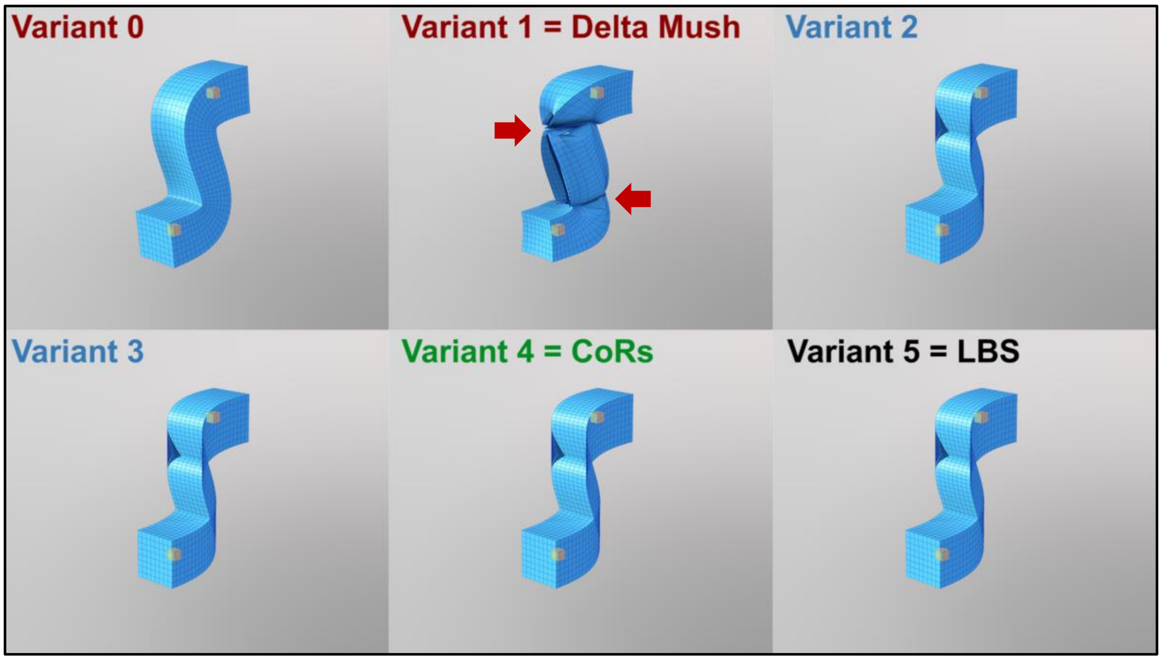
So, we propose some variants of the algorithm to cut some corners and improve the performance.
Here, variant 0 is the original SVD algorithm in the previous slide.
Variant 1 uses inverse transpose to approximate the closet rotation. This is similar to the way of transforming normal vector with LBS.
Variant 2 and 3 reduce the data storage and computation by using one scalar to represent the rotation (which is illustrated by a purple square). Instead of solving for the rotation, we just do rotation blending with this purple scalar weight. and we can either use quaternion (which is variant 2) or linear blending, which is variant 3.
Variant 4 further reduce the number of multi weight to 2 and variant 5 just use scalar weight.
[click]
Some variants are equivalent to previous skinning algorithms:
Variant 1 produces exactly the same result as the original Delta Mush, in which the local coordinate frame is represented by two tangent vectors.
Variant 4 here gives the explicit formula to compute the center of rotation for each vertex, which is equivalent to my SIGGRAPH 2016 paper.
And Variant 5 is the same as baking original Delta Mush to Linear Blend Skinning using skinning decomposition.



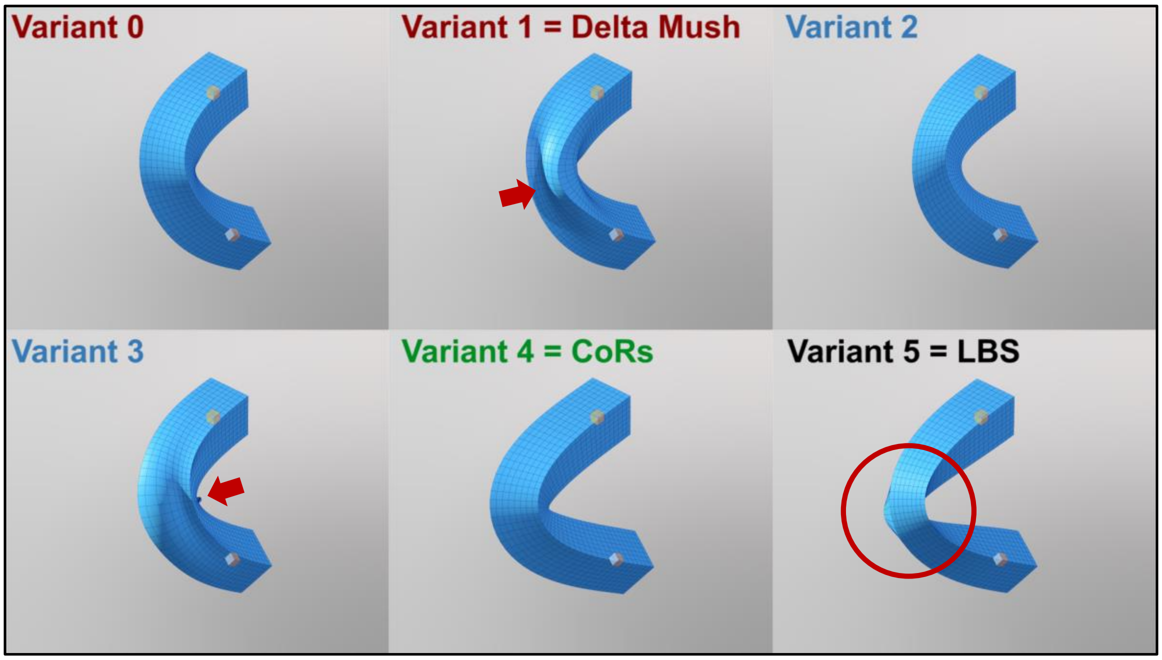
Here is an example to compare the deformation quality between different variants.



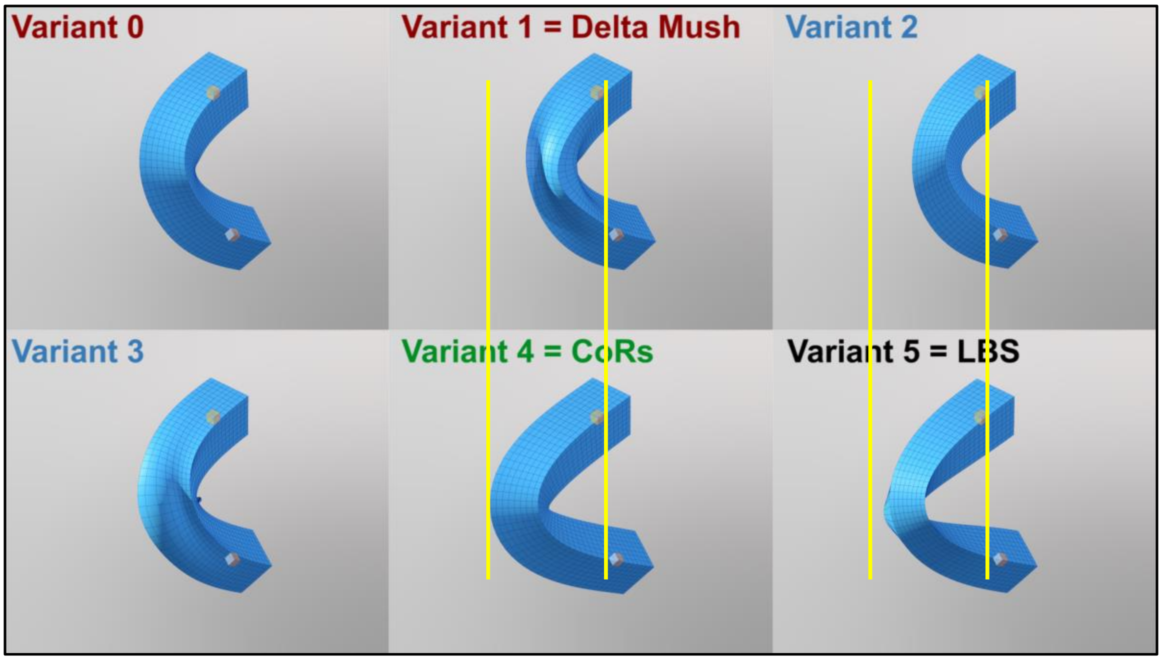
Now I will go back to some poses.
V1 /DM use inv trans, so there is distortion.
Note that in this pose, we only translate but not rotate the joint, so only v0 and v1, which use full 16 multi-weight can propagate this translate into local rotation.
V0 here show nice result, just like many indirect deformers like ARAP.
[click]
But remember than starting from v2, we only use 1 scalar for rotation part, so change in bone translations cannot propagate to the rotation part.



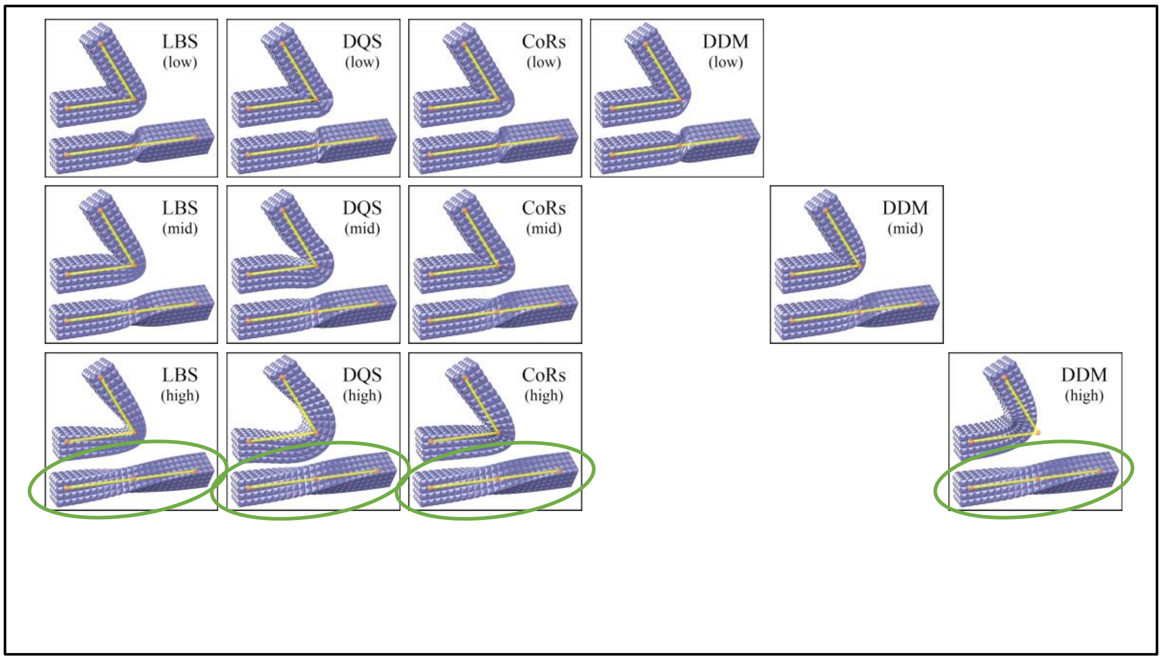
If we lift the joint further, some local transformation of v1 will be inverted.



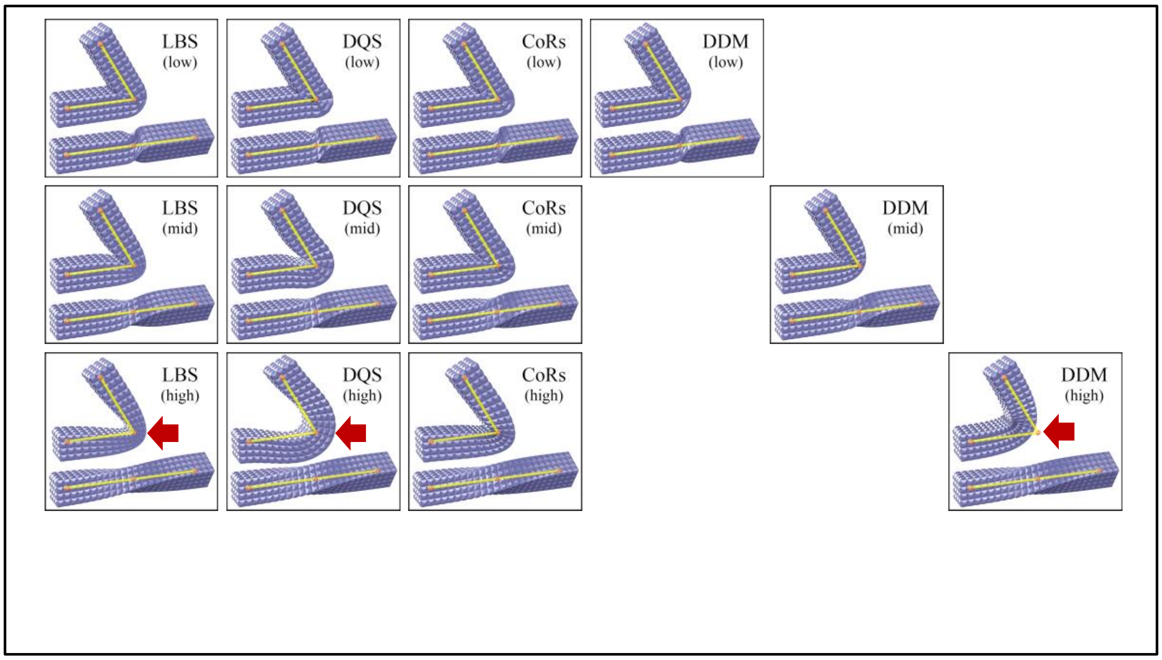
However, in practice, normally we control both rotation and translations. So the difference is less noticeable between variants.
In this case we still see some distortion of v1 here [click] and v3. [click]
We also notice that v5, which is LBS collapse here [click], that is the candy wrapper artifact.



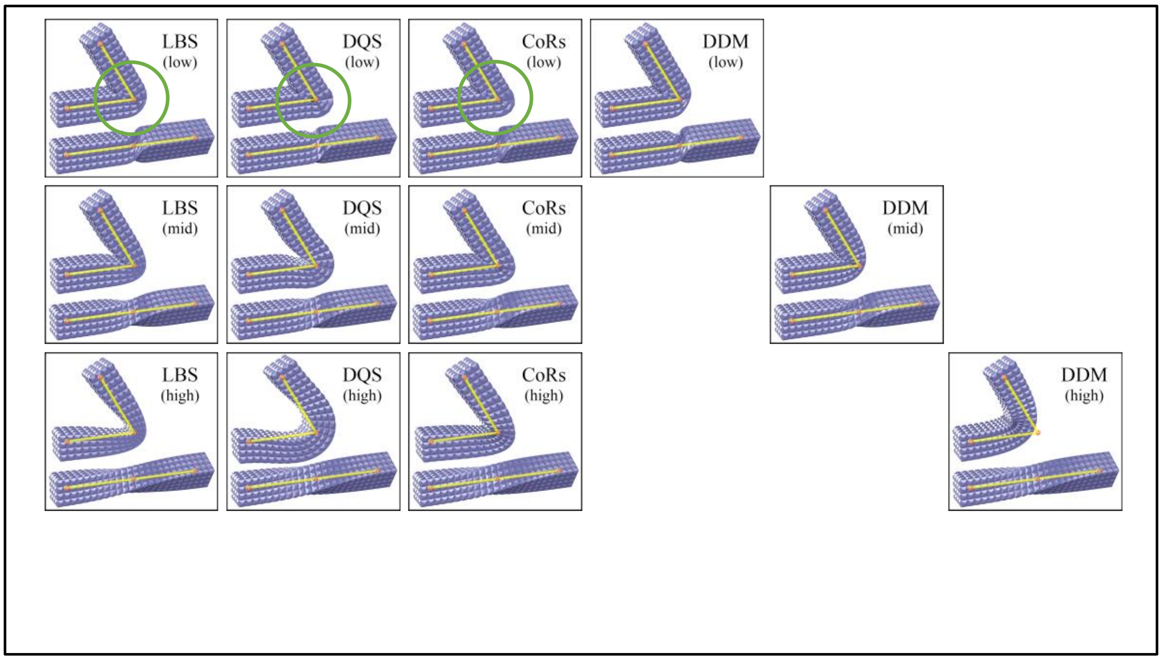
Another thing I want to point out is although the input bone transformations are the same, the profile that different variants bend the bar are different.
In this case there might be no explicit right or wrong solution but would be nice if we can have a control over that.



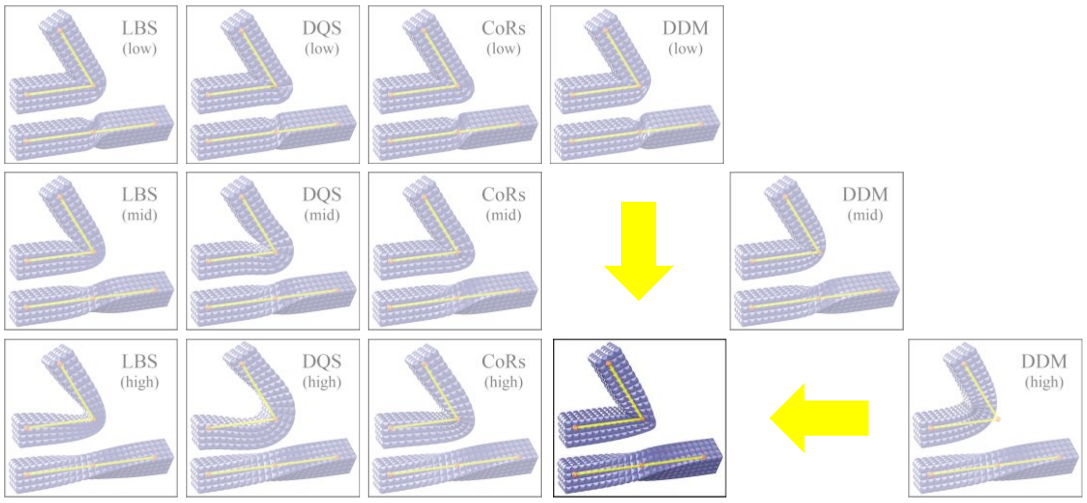
So we introduced an extension for that type of control.
Here is an example of bending and twisting a bar with different level of smoothness.
They are setup either by changing the skinning weights diffusions or tuning the number of Laplacian iterations with DDM.
We can see the bar twisted quite nicely with high smoothness, just like the skin sliding we have when twisting the arm.



However, the bending is not good. You can think of it doesn't look like the silhouette of the bent.

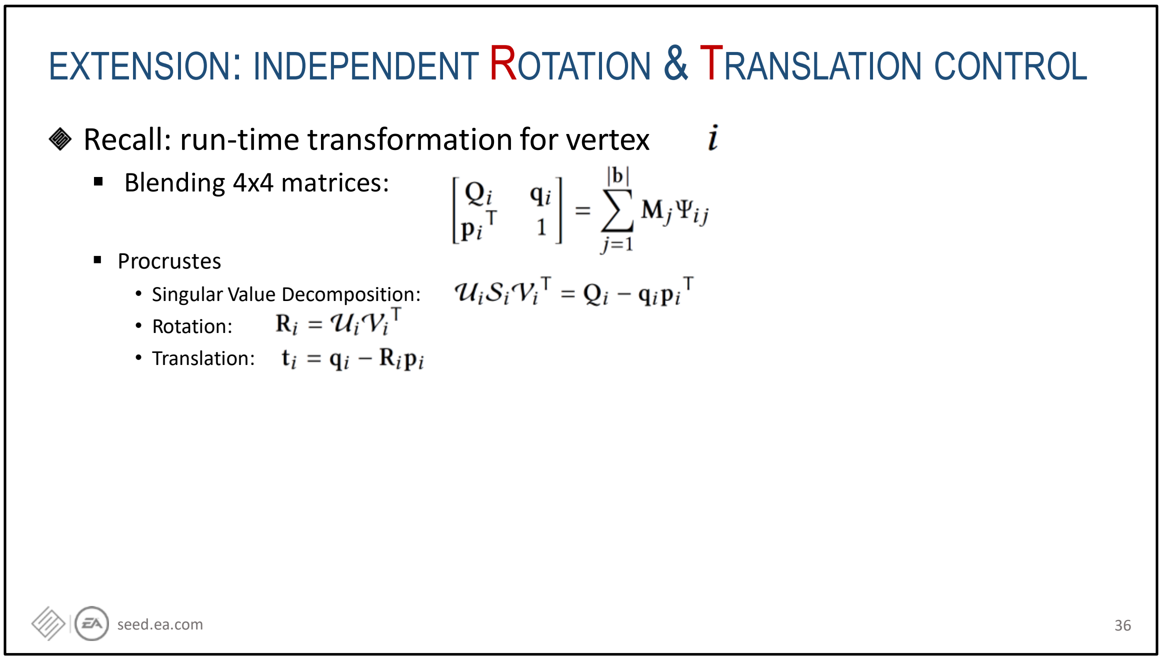


Indeed lower smoothness looks better.



EXTENSION: INDEPENDENT ROTATION & TRANSLATION CONTROL

So to bring the best of both, we extend our model to combine two different results as you can see here.
The twisting is smooth while the bending is rigid and looks more like an elbow.
This kind of behavior is not possible with previous methods, even with traditional Delta Mush.

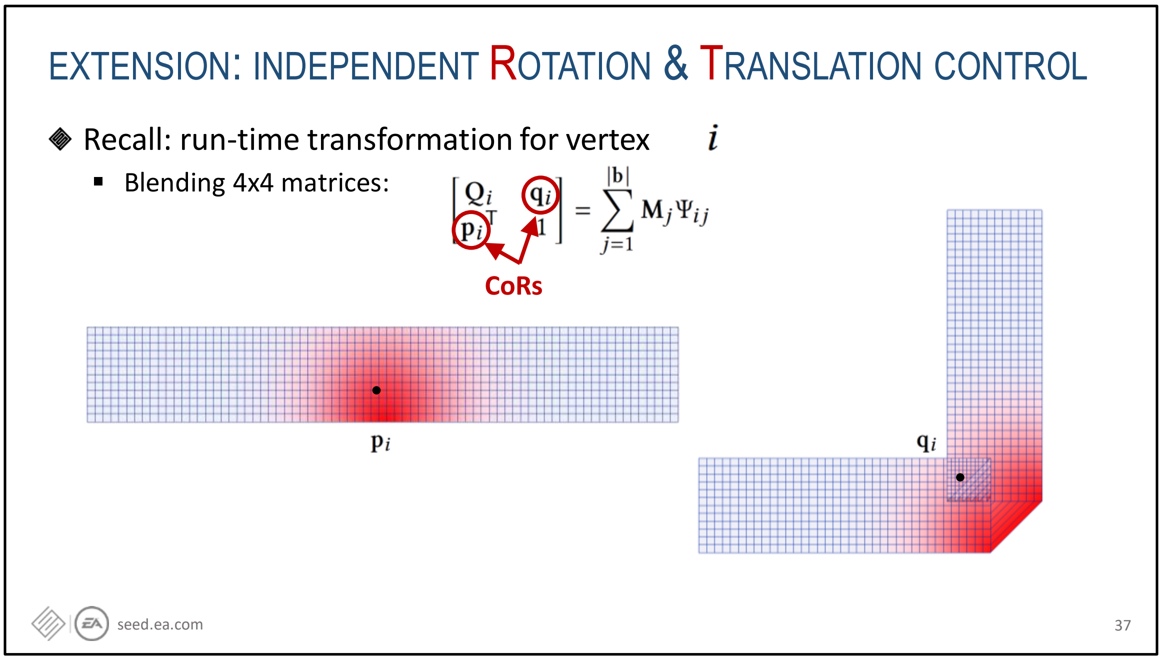


EXTENSION: INDEPENDENT ROTATION & TRANSLATION CONTROL

- Recall: run-time transformation for vertex i
  - Blending 4x4 matrices: $\begin{bmatrix} \mathbf{Q}_i & \mathbf{q}_i \\ \mathbf{p}_i^{\mathsf{T}} & 1 \end{bmatrix} = \sum_{j=1}^{|\mathbf{b}|} \mathbf{M}_j \Psi_{ij}$
  - Procrustes
    - Singular Value Decomposition: $\mathcal{U}_i \mathcal{S}_i \mathcal{V}_i^{\mathsf{T}} = \mathbf{Q}_i - \mathbf{q}_i \mathbf{p}_i^{\mathsf{T}}$
    - Rotation: $\mathbf{R}_i = \mathcal{U}_i \mathcal{V}_i^{\mathsf{T}}$
    - Translation: $\mathbf{t}_i = \mathbf{q}_i - \mathbf{R}_i \mathbf{p}_i$

seed.ea.com 36

We can do that with no extra cost at runtime. Let’s revisit the algorithm here.

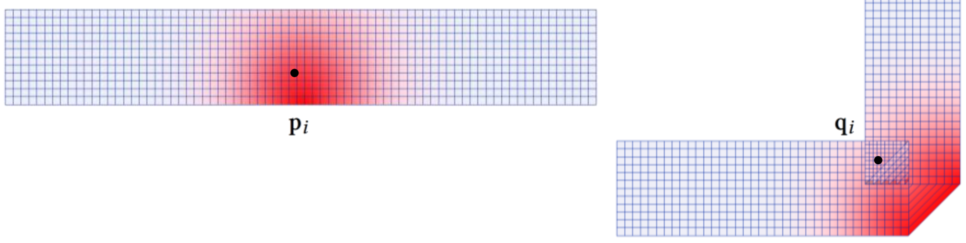


EXTENSION: INDEPENDENT ROTATION & TRANSLATION CONTROL

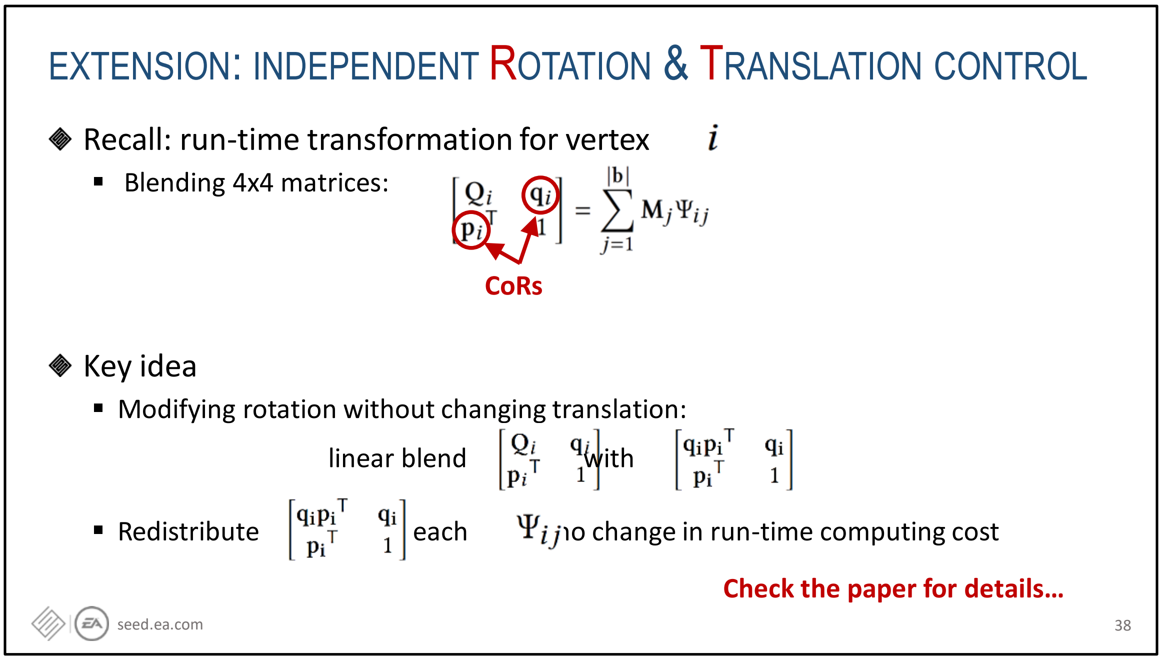
- Recall: run-time transformation for vertex i
  - Blending 4x4 matrices:

$$\begin{bmatrix} \mathbf{Q}_i & \mathbf{q}_i \\ \mathbf{p}_i^T & 1 \end{bmatrix} = \sum_{j=1}^{|\mathbf{b}|} \mathbf{M}_j \Psi_{ij}$$

CoRs



We found after blending transformations to get the 4x4 matrix, these two sub-blocks of the matrix are center of rotations of the Laplacian local patch.
p sub i corresponds to the rest pose and q sub i corresponds to the articulated pose.



Extension: Independent Rotation & Translation Control

- Recall: run-time transformation for vertex i
  - Blending 4x4 matrices: $\begin{bmatrix} \mathbf{Q}_i & \mathbf{q}_i \\ \mathbf{p}_i^T & 1 \end{bmatrix} = \sum_{j=1}^{|b|} \mathbf{M}_j \Psi_{ij}$

CoRs

- Key idea
  - Modifying rotation without changing translation:

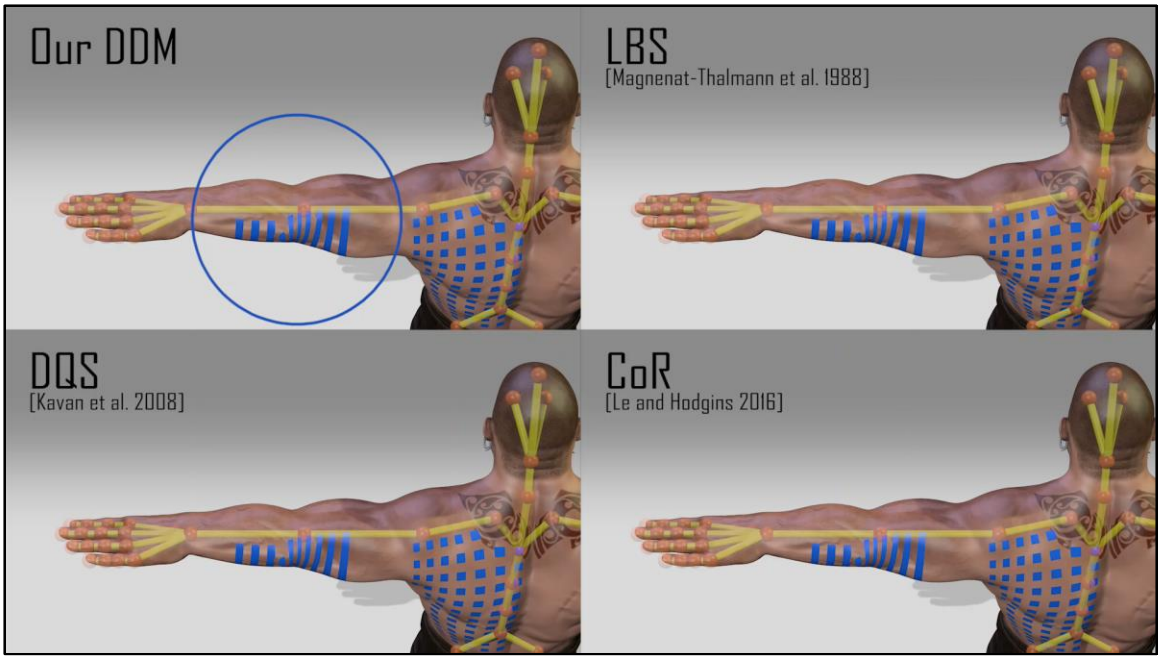
linear blend $\begin{bmatrix} \mathbf{Q}_i & \mathbf{q}_i \\ \mathbf{p}_i^T & 1 \end{bmatrix}$ with $\begin{bmatrix} \mathbf{q}_i \mathbf{p}_i^T & \mathbf{q}_i \\ \mathbf{p}_i^T & 1 \end{bmatrix}$

  - Redistribute $\begin{bmatrix} \mathbf{q}_i \mathbf{p}_i^T & \mathbf{q}_i \\ \mathbf{p}_i^T & 1 \end{bmatrix}$ each Ψ_{ij} no change in run-time computing cost

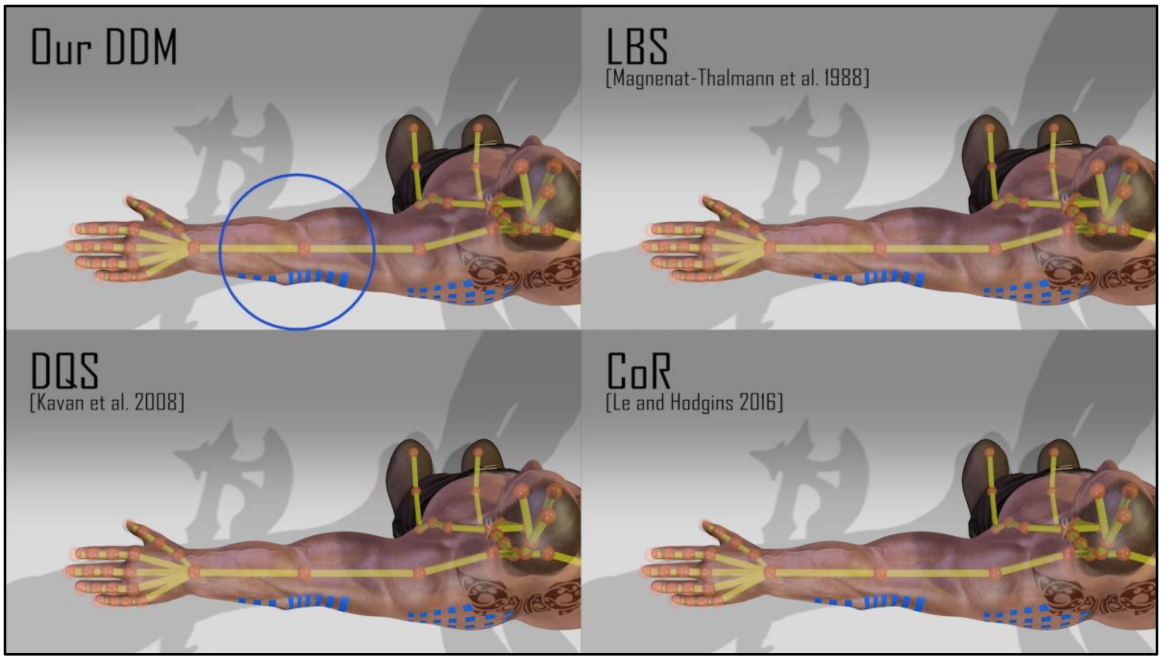
Check the paper for details…

seed.ea.com 38

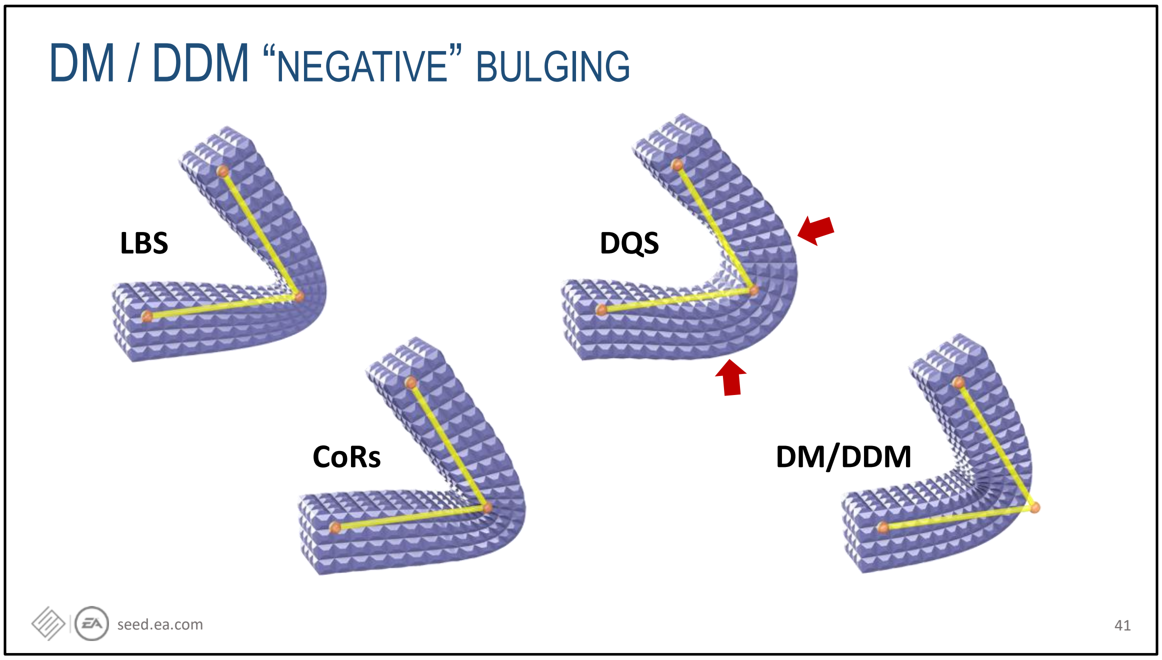
So if we use this q times p transpose to construct a new 3x3 sub matrix and blend it to the original, we can change the rotation matrix without changing the translation. Once we have the new matrix, we can redistribute them back to each of the multi weight Psi sub ij, so to the runtime algorithm, the changes will be transparent. And therefore, it does not increase run-time computing cost.
That’s the high level idea, please check the paper for more details.



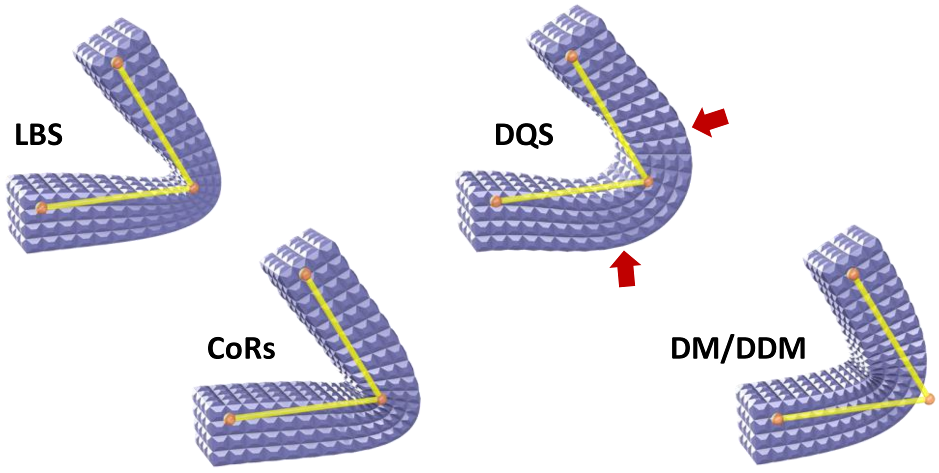
And now is an example of using the extension to setup the skin sliding.
So here you can see the skin does not slide much for other methods such as LBS, DQS, or CoR because of their skinning weights do not diffuse much further from the elbow joint.
And the reason is artist need to balance between smoothness for twisting and smoothness for bending.



Here is how the deformation looks with bending animation.
You can see that although our DDM has smoother skin sliding on twisting, it has sharper profile on bending, which better represents the inside rigid bone knuckle.
And you can still see some skin sliding here.
That’s all thank to the independent rotation and translation control.



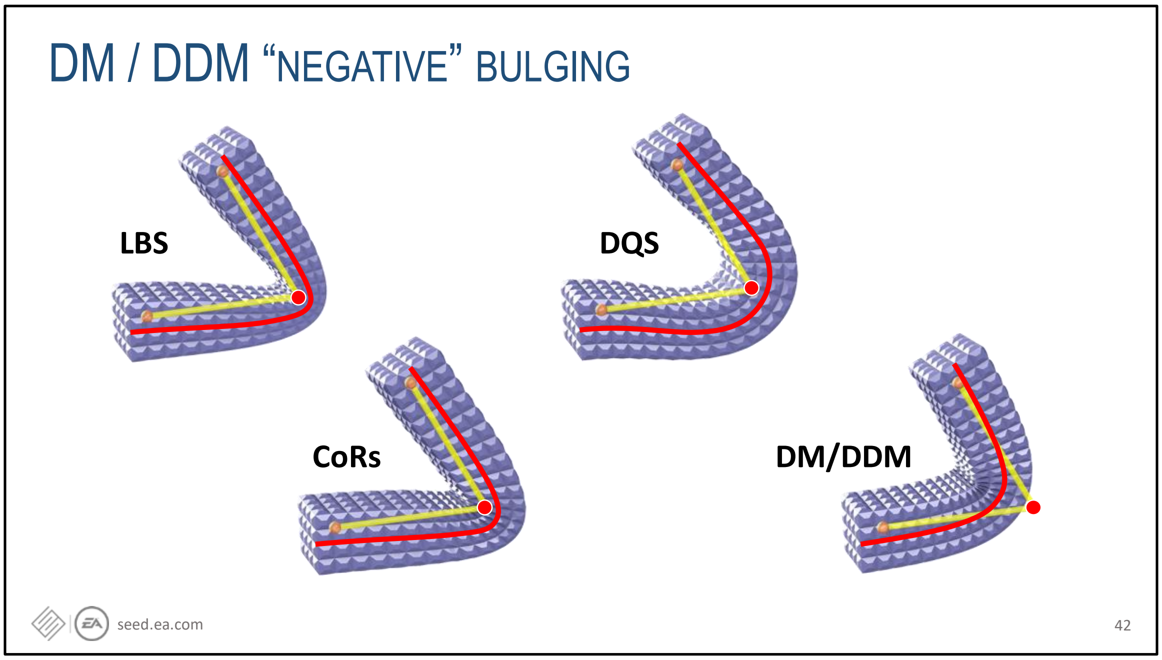
DM / DDM "NEGATIVE" BULGING



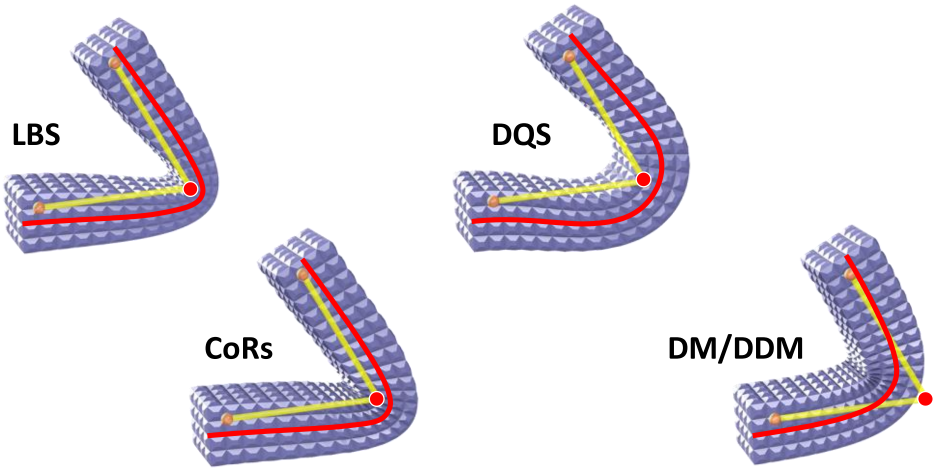
seed.ea.com

41

Another property that the independent control supports well for Delta Mush is to utilize the effect we called negative bulging.
[click]
So you might all familiar with the bulging artifact, noticeable with DQS.
And actually LBS and Optimized CoRs skinning also have bulging issue.



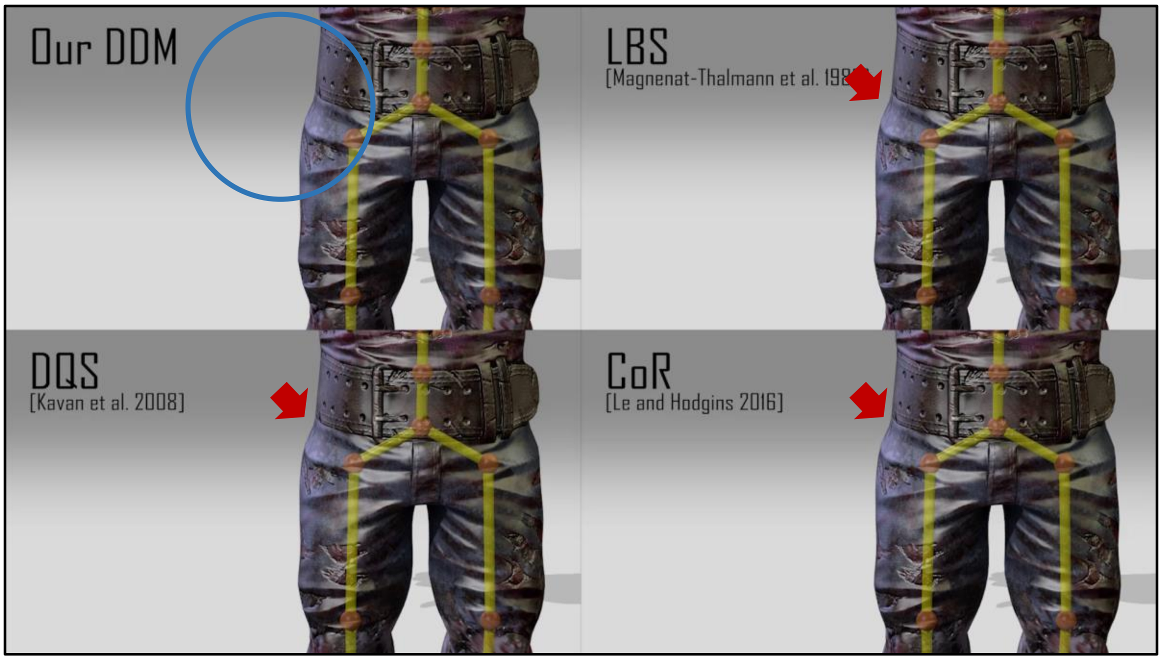
DM / DDM "NEGATIVE" BULGING



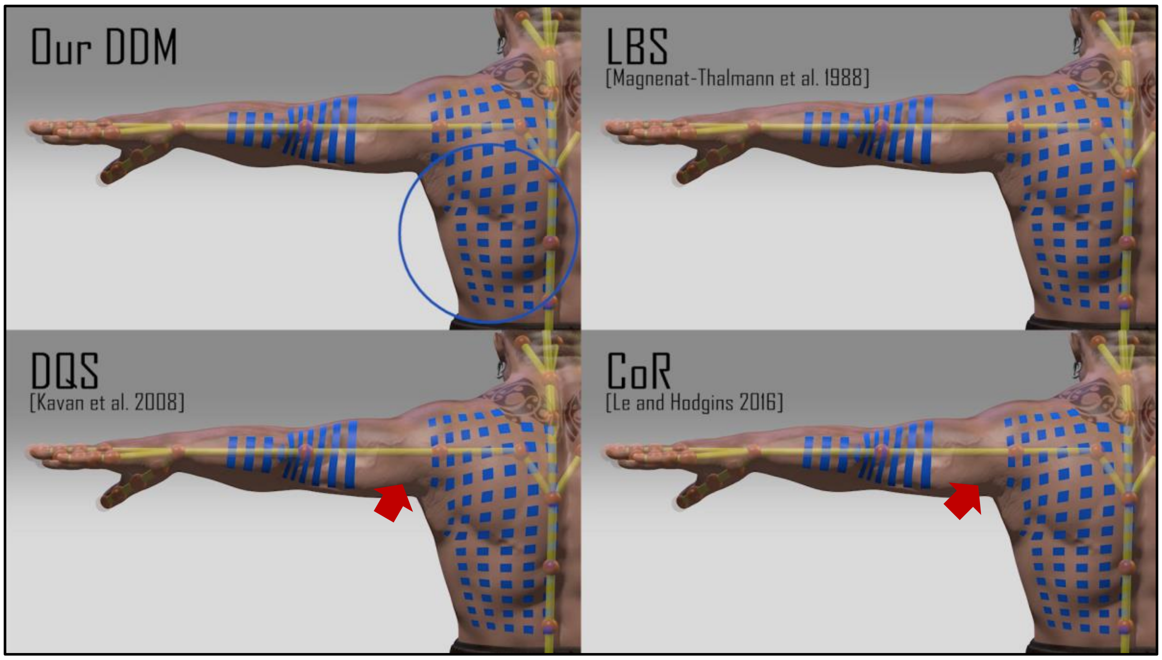
seed.ea.com

42

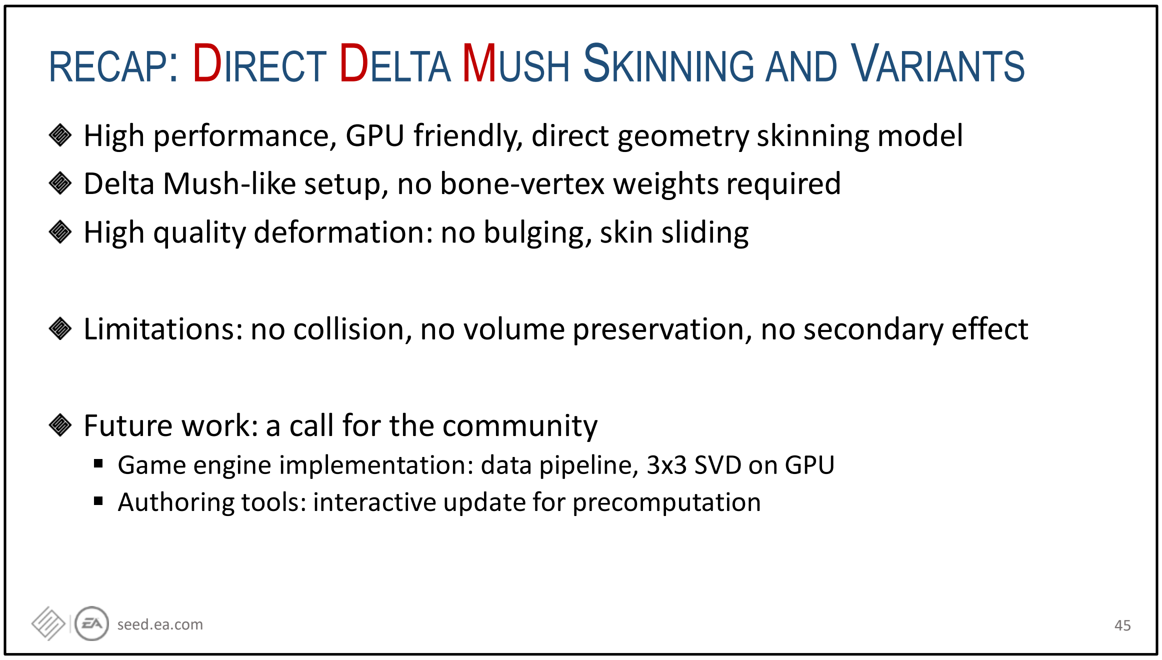
It is more noticeable if we draw the centerline of the side surface.
You can see the pattern here with previous skinning methods is the centerlines budge out of angle limited by two bones.
Only DM and DDM have this centerline curve inside.



This property is useful for the hip joint derformation here. Normally, artist need to put corrective shapes on top of traditional skinning models like LBS or DQS to achieve the same effect.



Here is the last example, I show skin sliding on the back.
Other methods, cannot have skin movement in this region because painting skinning weight of the arm down to this region will lead to more bulging.
You can see that even with influence of the arm stops very close to the shoulder, DQS and optimized CoR skinning still suffer with bulging.



RECAP: DIRECT DELTA MUSH SKINNING AND VARIANTS

- High performance, GPU friendly, direct geometry skinning model
- Delta Mush-like setup, no bone-vertex weights required
- High quality deformation: no bulging, skin sliding

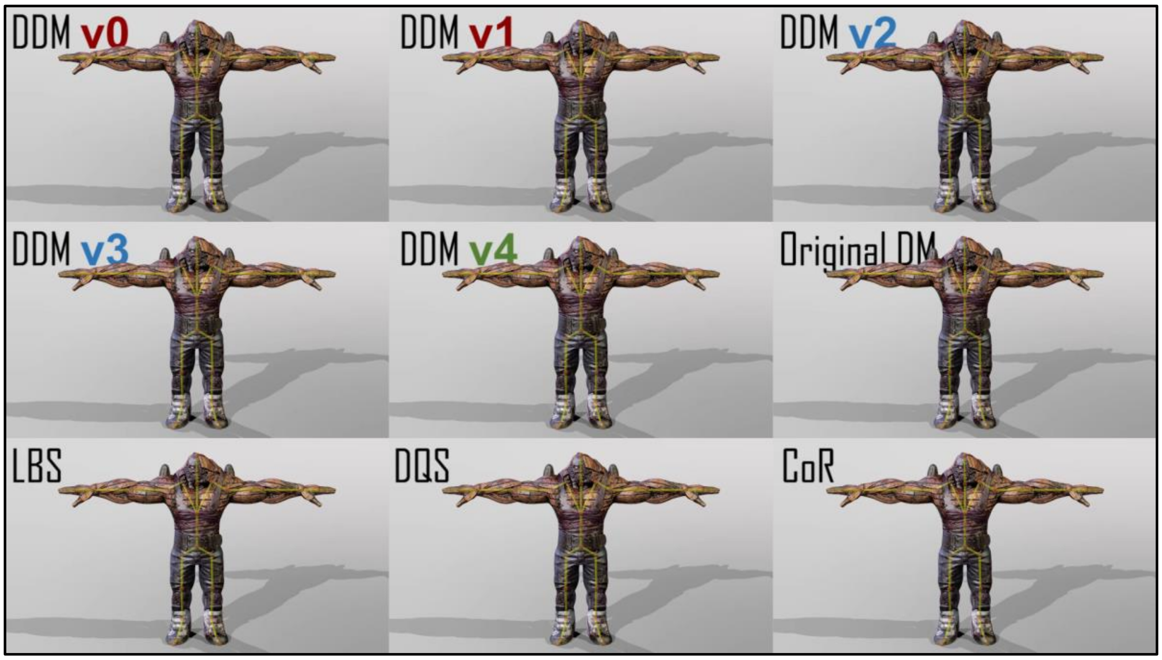
- Limitations: no collision, no volume preservation, no secondary effect

- Future work: a call for the community
  - Game engine implementation: data pipeline, 3x3 SVD on GPU
  - Authoring tools: interactive update for precomputation

seed.ea.com 45

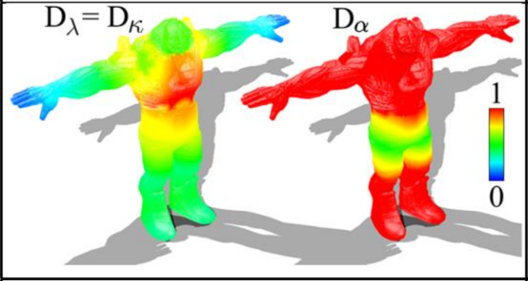
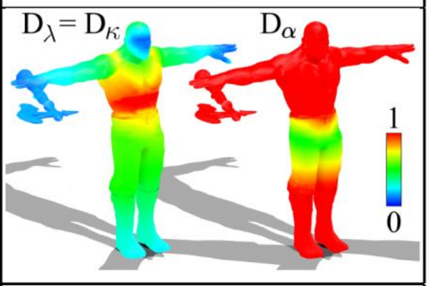
And to recap, we have made Delta Mush be a direct skinning model, which is suitable for high performance graphics.
We also did some extensions to further improve the deformation quality compared to the original DM model.
Specially, our model is the first direct method with no bulging and support skin sliding.
However, we have not solved many other problems such as collision, volume preservation, or secondary deformation effect.
We hope that this work can provide theoretical ground and inspiration for the community to further explore this direction. Especially for those who use and love Delta Mush.

To learn more details about this paper: http://binh.graphics/
Questions and comments could be addressed to the first author:
bbinh85@gmail.com



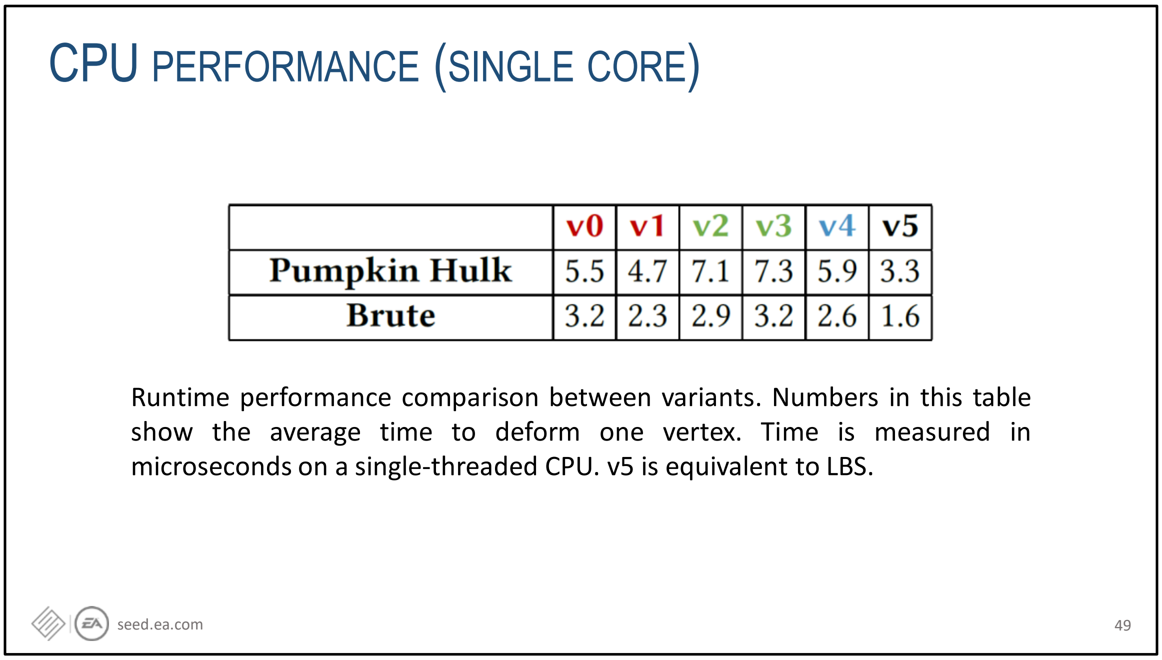
Comparison on test animation sequence.

Models and Setups

Pumpkin Hulk	Brute
	
$n = 20604$, $m = 65$	$n = 55886$, $m = 67$
Global parameters: $p = 20$, $\lambda = 20$, $\kappa = 1$, $\alpha = 0.9$	
precomputation time = 2.2s	precomputation time = 9.7s

seed.ea.com

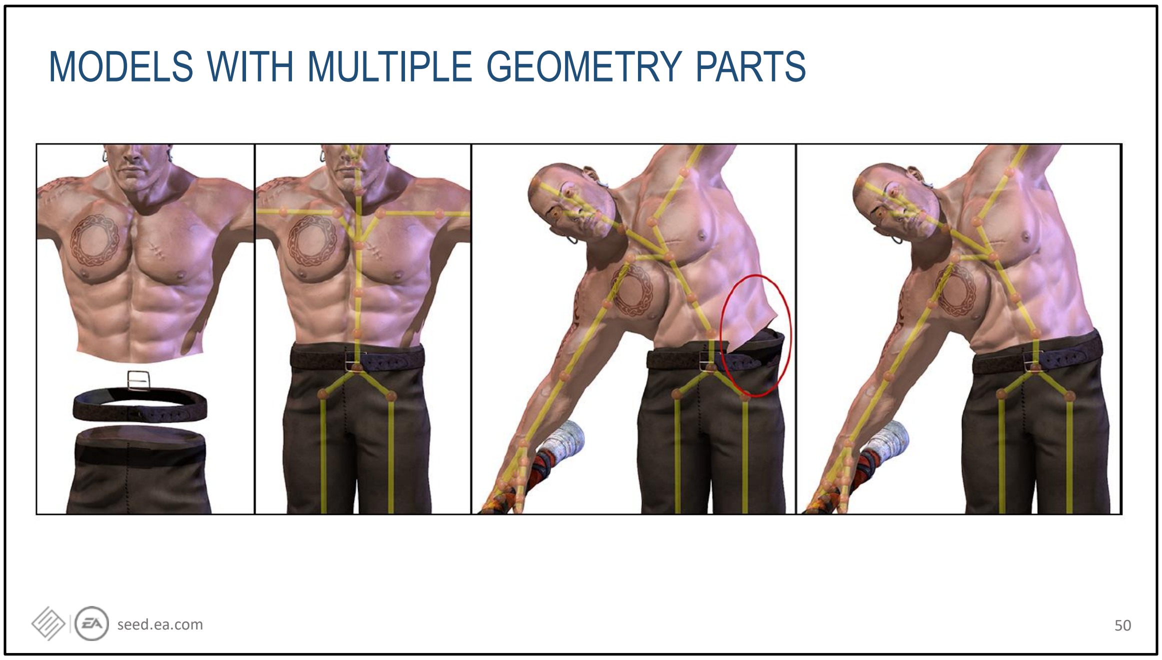
Table 2. The test models, parameters, and pre-computation time of our method. Test models are shown at their rest pose with color coded smoothing weights. n denotes the number of vertices, m denotes the number of bones. Per-vertex smoothing weights Dλ and Dκ are set proportional to the distance from the vertex to the bone to approximately represent the radius of each body part. Per-vertex blending weights Dα of vertices on the hip area are set to lower values than on other vertices.



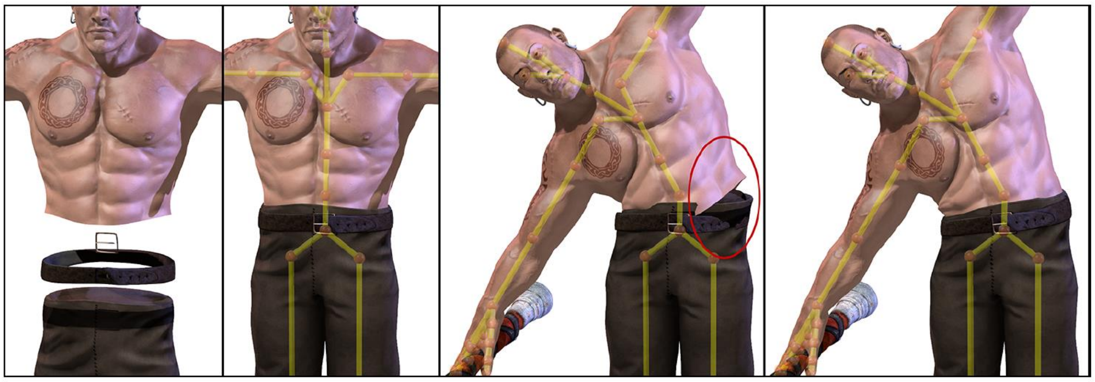
CPU PERFORMANCE (SINGLE CORE)

	v0	v1	v2	v3	v4	v5
Pumpkin Hulk	5.5	4.7	7.1	7.3	5.9	3.3
Brute	3.2	2.3	2.9	3.2	2.6	1.6

Runtime performance comparison between variants. Numbers in this table show the average time to deform one vertex. Time is measured in microseconds on a single-threaded CPU. v5 is equivalent to LBS.



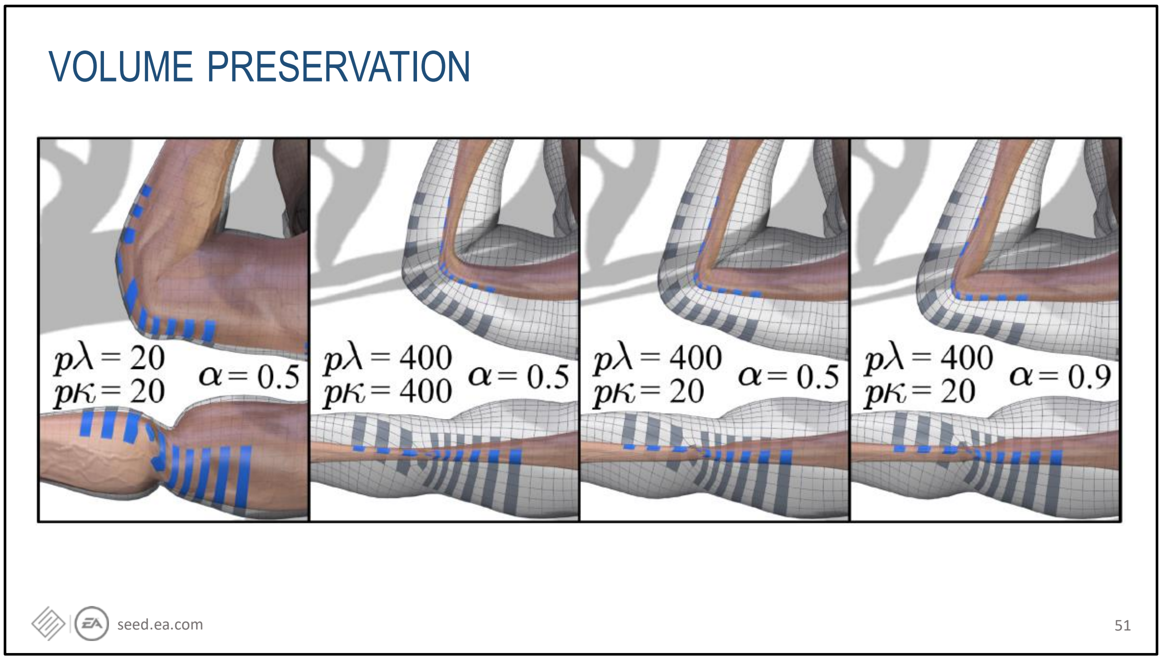
MODELS WITH MULTIPLE GEOMETRY PARTS



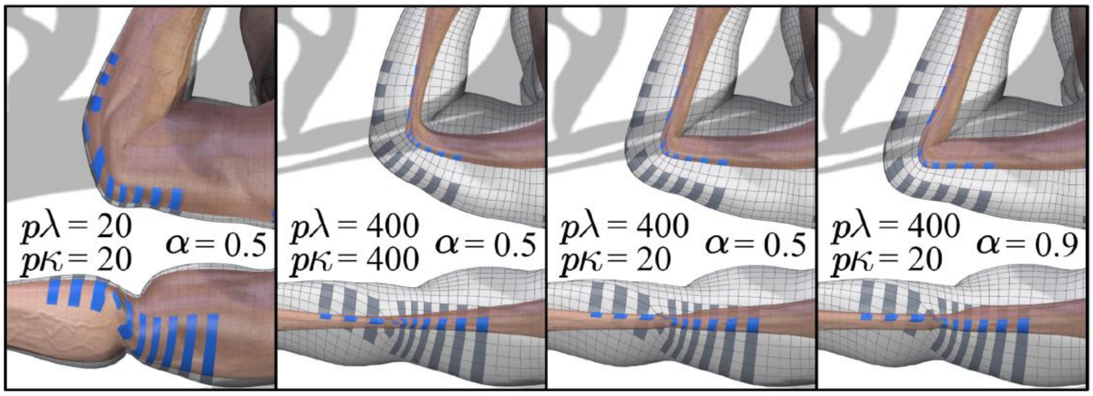
seed.ea.com

50

Fig. 6. Our trivial solution to handle models with multiple geometry parts. From left to right: (1) visualization of multiple parts of our character model, (2) the model at rest pose, (3) applying per-part mesh Laplacians breaks the deformation at boundaries of parts, (4) our fix by adding entries to the Laplacian matrix connecting close vertices associated to the same bone.



VOLUME PRESERVATION



seed.ea.com

51

Fig. 11. Volume preservation behavior of the model (gray wireframed) depends on the volume of the mush (color shaded) at the rest pose. Leftmost: with low smoothness pλ, the mush does not shrink much, but some of this remaining volume can be lost with deformation. Middle and right: with high smoothness pλ, the mush shrinks to a rod-like shape with near zero volume and this cannot be further reduced by deformation. Bending deformation can be controlled by changing the translation smoothness pκ and its blending weight α.